

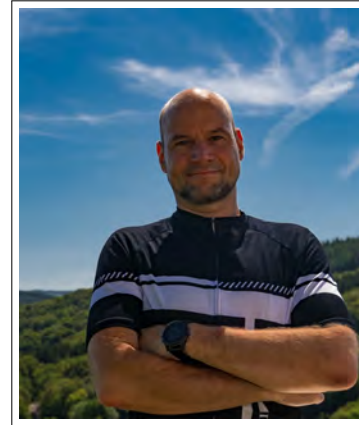
Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

Meet the Lecturers ...



Prof. Dr. Ingo Scholtes



Dr. Anatol Wegner

Notes:

- **Lecture L01: Motivation** 22.10.2024
- **Educational objective:** We give an overview of the history of computing and motivate challenges at the intersection of computer science and law. We discuss organizational aspects and give an outline of the course.
 - Introducing the Chair
 - A Brief History of Computer Science
 - Informatics, Law, and Humanities
 - Course Overview and Organizational Issues

Notes:

Who am I?

1/2

- ▶ **diploma in Computer Science** from University of Trier (2005)
- ▶ designed **large-scale distributed Event Monitoring system** of ATLAS detector at CERN (2004–2008)
- ▶ **doctorate degree** in the area of massive-scale distributed systems from University of Trier (2011)



the ATLAS detector, recording data on 40 million particle-particle collisions per second at 2 MB per collision

image credit: CERN

Who am I?

2/2

- ▶ **postdoc/lecturer** at ETH Zürich (2011 – 2018)
- ▶ **SNSF Professor for Data Analytics** at University of Zurich (2018 – 2024)
- ▶ **Full Professor of Data Analytics** at Bergische Universität Wuppertal (2019 – 2021)
- ▶ **Full Pofessor of Machine Learning for Complex Networks** at JMU (since 2021)



Festung Marienberg at night

image credit: Ingo Scholtes

Notes:

Notes:

Chair of Machine Learning for Complex Networks



Franziska
Heeg



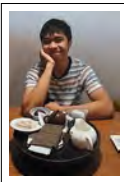
Moritz
Lampert



Jan
von Pichowski



Lisi
Qarkaxhija



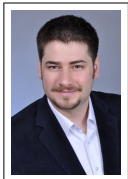
Chester
Tan



Dr. Christopher
Blöcker



Dr. Vincenzo
Perri



Dr. Anatol
Wegner



Prof. Dr. Ingo
Scholtes

Research interests



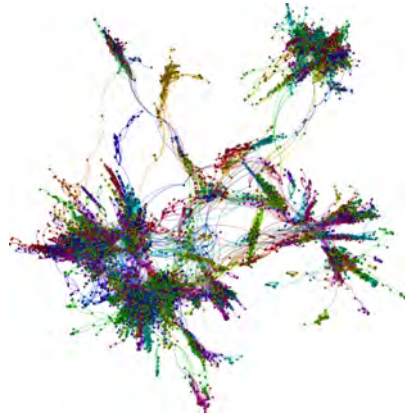
Notes:

Notes:

Teaching portfolio

Lecture portfolio

- ▶ BSc lecture **Combinatorial and Sequential Circuits**
→ University of Trier
- ▶ BSc lecture **Computer Architecture and Assembler Programming**
→ University of Applied Sciences Trier
- ▶ BSc lecture **Computer Networks**
→ University of Trier
- ▶ VWA lecture **Web Programming**
→ VWA Trier
- ▶ BSc lecture **Information Systems for eCommerce**
→ KIT
- ▶ BSc lecture **Introduction to Databases**
→ University of Wuppertal
- ▶ BSc lecture **Algorithms for AI & DS II**
→ JMU
- ▶ MSc lecture **Introduction to Data Science**
→ University of Wuppertal
- ▶ MSc lecture **Statistical Network Analysis**
→ ETH Zürich, University of Zurich, JMU
- ▶ MSc lecture **Machine Learning for Complex Networks**
→ JMU
- ▶ MSc lecture **Introduction to CS for Students from All Faculties**
→ JMU



dependency network of a complex software system

Who are you?

Questions

- ▶ what is your **background**?
- ▶ did you have any **prior contact to Informatics or Computer Science**?
- ▶ what **specific interests or needs** do you have based on your professional practice?
- ▶ what are **your expectations** for this course?



DALL-E generated image using the prompt "generate an image of an audience of computer science, law, and humanities students"

Notes:

Notes:

What is a computer?

1/3



Macbook M1 Pro notebook computer

image credit: Wikipedia, VladIsLoveW, CC-BY-SA
Lecture 01: Motivation

Ingo Scholtes

Introduction to Informatics

October 22, 2024

8

What is a computer?

2/3



Apple S2 System-on-a-Chip (approx. 2 gigaFLOPS)

1 gigaFLOP = 1 billion = 10^9 floating point operations per second

image credit: Wikipedia, CC-0

Ingo Scholtes

Introduction to Informatics

Lecture 01: Motivation

October 22, 2024

9

Notes:

Notes:

What is a computer?

3/3



xAI Colossus GPU Cluster, the world's most powerful AI supercomputer
100,000 H100 GPUs (USD 25,000 each) with approx. 6 exaFLOPS

1 exaFLOP = 1 quintillion = 10^{18} floating point operations per second

image credit: xAI, fair use

From human to machine calculations ...

- ▶ 17th – 20th century: **computer = occupation**
 - ▶ humans **performing tedious calculations**
 - ▶ applications in astronomy, navigation, engineering, military
- ▶ 19th century: **mechanical computers**
 - ▶ 1820s: Babbage's difference engine
 - ▶ implements "divided differences" method to **tabulate logarithms and trigonometric functions** → C Babbage, 1824
- ▶ late 19th century: **desktop calculators**
 - ▶ 1851: Thomas Colmar's Arithmometer
 - ▶ 1879: National Cash Register Co.
 - ▶ 1911: Marchant calculator
 - ▶ 1920: Hamann Manus calculators



human computer at NASA, 1952

image credit: NASA, Public Domain

limitation

early mechanical computers were limited to calculation of **fixed set of arithmetic operations**

Notes:

Notes:

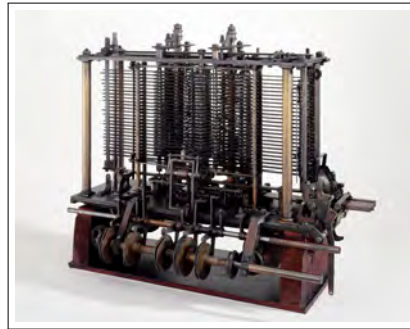
Automating Computing

▶ programmable mechanical computers

- ▶ Babbage's Analytical Engine (1837)
- ▶ supported all four arithmetic operations and comparisons
- ▶ 55,000 parts, 19 meters long, driven by steam engine (but never completed)
- ▶ programmable via **punched cards**, i.e. "software"

▶ electromechanical computers

- ▶ Konrad Zuse's Z3 (1941)
- ▶ first **digital programmable** computer
- ▶ programs stored on punched tape
- ▶ 2,000 relays, 30,000 cables, 1 ton, 4 kW
- ▶ approx. five operations per second



Part of Babbage's Analytical Engine

image credit: Wikipedia, User Mrjohncumings, CC-BY-SA

Electronic Computers

▶ ENIAC (1945)

- ▶ "Electronic Numerical Integrator and Computer"
- ▶ programmable by changing cabling
- ▶ 17,468 vacuum tubes, 27 tons
- ▶ USD 7.5 mio, 175 kW
- ▶ 20 – 400 operations per second
- ▶ built for US Army to calculate artillery firing tables

▶ UNIVAC I (1951)

- ▶ "UNIVersal Automatic Computer"
- ▶ first **commercial electronic programmable** computer
- ▶ 5,000 vacuum tubes, 13 tons
- ▶ USD 7 mio, 125 kW
- ▶ 1,900 operations per second
- ▶ produced until 1986 (!)



ENIAC ca. 1947

image credit: US Army, public domain

Notes:

Notes:

Computing pioneers: Alan Turing

- ▶ **Enigma machine**: electromechanical device used by German military to encrypt/decrypt communication during WW2
- ▶ **Bletchley Park Bombe**: electromechanical machine developed to break encryption of enigma machine
- ▶ mathematician **Alan Turing** played key role in design of Bombe at Bletchley Park
→ A Hodges: Alan Turing: The Enigma
- ▶ formalized **concept of “computability”** that became foundation of computer science

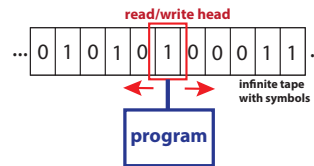
definition

A **Turing machine** is a mathematical model that is used to formalize which problems we consider to be “computable”. We currently consider a problem to be in principle solvable by a computer, if it can be solved by a Turing machine in a finite number of steps. → A Turing, 1937



Alan Turing, 1912 – 1954

image credit: Wikipedia, public domain



Computing pioneers: John von Neumann

- ▶ physicist involved in Manhattan project at Los Alamos Laboratory
- ▶ engaged in programming of ENIAC and **design of successor EDVAC** (1950s)
- ▶ formalized concept of **stored-program computers**, which store both data and instructions (program) in memory

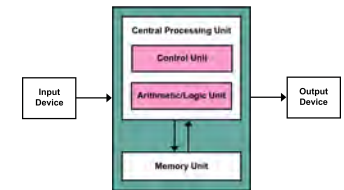
Von Neumann architecture → J von Neumann, 1945

1. **input and output**
 2. **memory**, holding data and instructions
 3. **control unit**, which decodes instructions and controls their execution
 4. **arithmetic and logical unit**
- ▶ basis for (almost) all modern computers



John von Neumann, 1903 – 1957

image credit: Los Alamos National Security, LLC

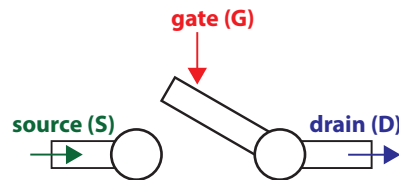


Notes:

Notes:

From Switches to Logical Operations

- ▶ electronic computers are fundamentally based on **switches** that implement **logical operations**
- ▶ gate controls whether electrons flow or not between source and drain (digital signal)
- ▶ we can use switches to implement **logical and arithmetic operations** → Lecture 02
- ▶ let us encode the **binary states** of source, gate, and drain as 0 (off) and 1 (on)
- ▶ can you describe the binary state of drain based on the states of source and gate?



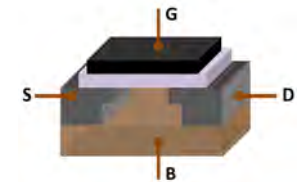
S	G	D
0	0	0
0	1	0
1	0	0
1	1	1

Electronic Switches: Relay, Tube, Transistor

- ▶ **relay** (1840)
 - ▶ **electromagnetic** switch
 - ▶ patented by Samuel Morse
 - ▶ used in Zuse Z3
- ▶ **triode vacuum tube** (1906)
 - ▶ **thermionic** switch
 - ▶ patented by Robert von Lieben
 - ▶ used in UNIVAC
- ▶ **transistor** (1947)
 - ▶ **semiconductor** switch
 - ▶ discovered by Bardeen, Brattain, Shockley (awarded Nobel prize in physics in 1956)
 - ▶ no mechanics, no heated tube
 - ▶ small, low power, cheap, mass producible
 - ▶ foundation of all modern electronics



Replica of first transistor

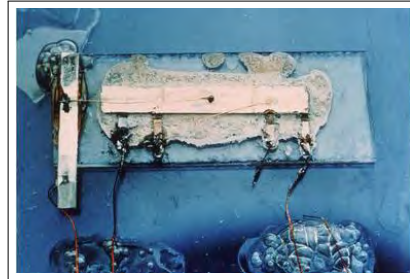


Notes:

Notes:

Integrated Circuit and Microprocessor

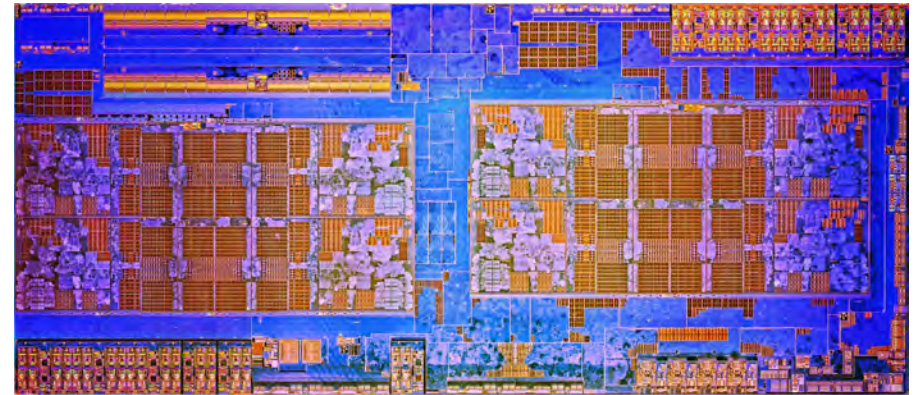
- ▶ **integrated circuit** (1958)
 - ▶ **multiple transistors on one chip**
 - ▶ developed by Jack Kilby (Nobel prize in physics in 2000)
 - ▶ complex layouts of transistors (switches) can be lithographically “printed”
 - ▶ basis for **miniaturization of computing hardware**
- ▶ **Intel 4004 microprocessor** (1971)
 - ▶ **first commercial microprocessor**, i.e. control, arithmetic, and logic unit integrated on a single chip
 - ▶ 2,250 transistors on 12 mm^2
 - ▶ approx. 1,100 arithmetic operations per second
 - ▶ approx. 450 USD, 12 mm^2 , 1 Watt



First integrated circuit (1958)

image credit: Texas Instruments, Fair Use

A Modern microprocessor



AMD Ryzen 7 1800X CPU with two CPU cores, each featuring four processing units
9,000,000,000 transistors on 200 mm^2

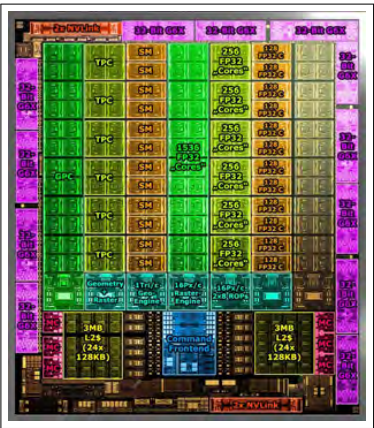
image credit: AMD

Notes:

Notes:

Modern Computing Hardware

- **Central Processing Unit (CPU)**
 - multiple processor cores integrated on single chip
 - 200 mm², 10 billion transistors
 - EUR 1000, 100 Watts
 - tens of **billions of operations** per second
- **Graphics Processing Unit (GPU)**
 - specialized hardware for vector calculations (and more)
 - used to **accelerate graphics processing and machine learning**
 - 600 mm², 75 billion transistors
 - EUR 1200, 450 Watts
 - tens of **trillions of arithmetic operations** per second



NVidia GA102 GPU

image credit: Tom's Hardware

Notes:

Moore's Law: The number of transistors on microchips doubles every two years. Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing - such as processing speed or the price of computers.

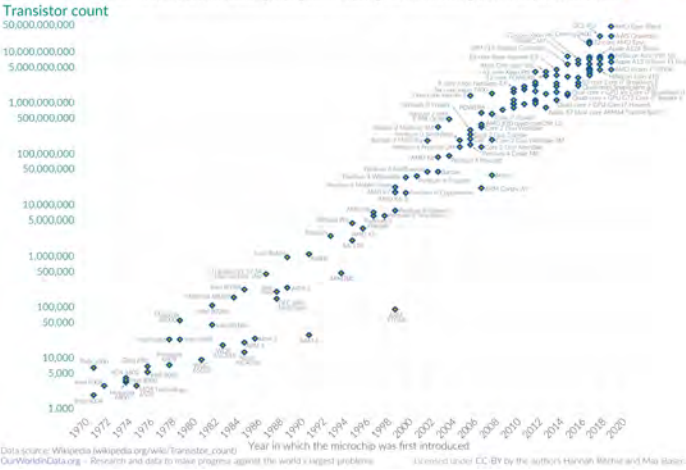


image credit: ourworldindata.org using 40 Years of Microprocessor Trend Data from Karl Rupp

Notes:

The age of algorithms

definition

An **algorithm** is a sequence of precisely defined (mathematical) instructions that must be executed to solve a given problem.

1	0	5	7
2	8 ₁	9	2
3	9	4	9

example

The **pencil-and-paper method** to add two decimal numbers is a simple example for an algorithm

- ▶ term “algorithm” goes back to “Book of Indian computation” by Muḥammad ibn Mūsā **al-Khwārizmī** (latinized: algorismi)
- ▶ in the 19th century **human computers** executed “algorithms” to solve specific problems (e.g. filling ballistic tables)
- ▶ with the rise of **automatic computers**, humans became “**programmers**” writing algorithms executed by machines
- ▶ proposing an algorithm to compute Bernoulli numbers with **Babbage’s Analytical Engine**, Ada Countess of Lovelace is considered **first “programmer”**

From human to machine “intelligence”

- ▶ since 1940s: **artificial neural networks**
 - ▶ Alan Turing: artificial neural network (1948) and Turing test (1950)
 - ▶ Donald Hebb: model for neural plasticity (1949)
 - ▶ John Hopfield: Hopfield networks (1982)
 - ▶ Rumelhart, Williams, Hinton: backpropagation algorithm (1986)
- ▶ 1960s & 1970s: first **AI hype**
 - ▶ 1954: Georgetown demonstration of machine translation
 - ▶ cold war: huge funding programs for AI created
- ▶ **skepticism** arises
 - ▶ ALPAC Report for US National Research Council (1966)
 - ▶ Marvin Minsky & Seymour Papert: “Perceptrons: An Introduction to Computational Geometry” (1969)
 - ▶ James Lighthill: “Artificial Intelligence: A General Survey” (1973)



John Hopfield
born 1933



Geoffrey Hinton
born 1947

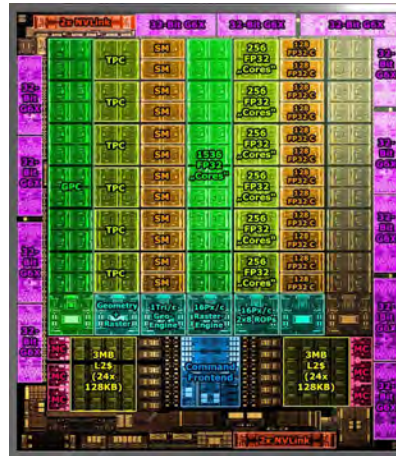
Nobel prize in physics in 2024

Notes:

Notes:

The AI Winter

- ▶ 1980s: **AI winter**
 - ▶ drastic cut-back of public funding and investments in artificial intelligence
 - ▶ field moves towards simpler “expert systems”
 - ▶ knowledgebases + automated reasoning
 - ▶ limited success due to difficult data acquisition
- ▶ 2010s: **AI hype reloaded**
 - ▶ 2000s: deep learning
 - ▶ powerful CPUs/GPUs: “cheap” learning
 - ▶ inflated(?) expectations in industry/media
- ▶ learning machine \neq intelligent machine



Nvidia G102 GPU
28.3 billion transistors
40 TeraFLOPs

The information age

- ▶ 1994: World Wide Web
 - ▶ Amazon (1994)
 - ▶ Google (1998)
 - ▶ Alibaba (1999)
 - ▶ Facebook (2004)
- ▶ up to 2003: humanity has produced approx. 1 billion gigabyte (= 1 exabyte) of data
- ▶ 2007: rise of **mobile devices**
- ▶ 2024: data everywhere
 - ▶ > 3 billion smartphones
 - ▶ 1 exabyte of data produced every 48 hours
 - ▶ top 10 Web companies > 800 billion annual revenue
 - ▶ data = “digital shadow” of humanity



result of DALL-E image generation for prompt
“Please generate an image that represents big data.”

image credit: www.bing.com

Notes:

Notes:

LLMs and Generative AI

- ▶ 2020s: boom of **generative AI**
 - ▶ rise of **Large Language Models** (LLMs) like GPT4 or LLaMA
 - ▶ **text-to-image models** DALL-E or Stable Diffusion
 - ▶ **text-to-music models** like MusicLM
 - ▶ **text-to-video models** like Runway or SORA
- ▶ fuelled by **massive volume of online training data** and **powerful hardware for deep neural networks**
- ▶ **new societal questions** emerge
 - ▶ intellectual property of generated texts, images, videos?
 - ▶ transformation of online content creation?
 - ▶ a new industrial revolution?
 - ▶ how to detect and counter deep fakes?
 - ▶ hints of “artificial general intelligence” (AGI) in LLMs?



image generated by Stable Diffusion using prompt “a photograph of an astronaut riding a horse”

[image credit](#): Wikimedia commons, public domain

Notes:

Informatics and Humanities

What is Informatics?

“Informatics is the study of the behavior and structure of [systems] that generate, store, process and then present information; it is basically **the science of information**. The field takes into consideration the **interaction between information systems and [users]**, as well as the construction of the interfaces between the two [...]”

→ [Techopedia](#)

What are Humanities?

“Humanities are academic disciplines that study **aspects of human society and culture**, including certain **fundamental questions asked by humans**. Today, the humanities are [...] frequently defined as any fields of study outside of natural sciences, social sciences, formal sciences (like mathematics), and applied sciences [...]”

→ [Wikipedia](#)

Informatics and Humanities

Lying at the intersection of informatics and humanities, the field of **digital humanities** covers the systematic generation and use of large digital resources on cultural data as well as the use of digital tools and computational methods to store and analyze them at large scale.

Notes:

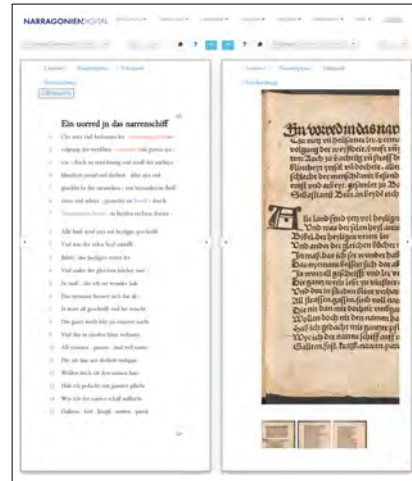
Example: Digital Editions

use case

Use of data storage and database technologies to permanently and securely archive cultural heritage and to make it accessible and searchable for researchers in the humanities.

exemplary questions for our course

1. How can we digitally encode data?
2. How can we permanently and securely store data and documents?
3. How can we efficiently query records from a database?



Example: Layout analysis and text recognition

use case

Use of machine learning and computer vision algorithms to automatically analyze the layout and text contents of historical documents.

exemplary questions for our course

1. What are the basic principles of machine learning?
2. What is the difference between supervised and unsupervised learning?
3. How can we use machine learning to automatically classify scanned documents and recognize text in image data?

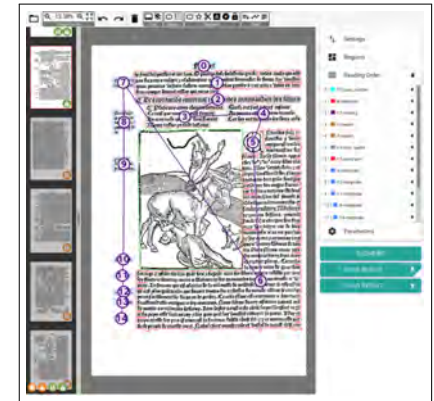


image credit: LAREX Layout analysis tool, developed at University of Würzburg

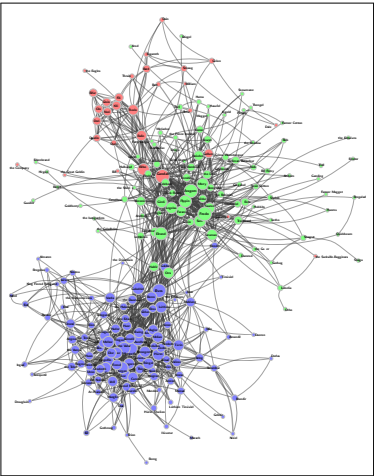
Notes:

Notes:

Example: Computational Literary Studies

exemplary case

Use of text mining, word embedding, network analysis, and deep learning techniques to analyze character co-occurrence networks in Tolkien's Legendarium.



exemplary questions for our course

- 1. How can we process large corpora of texts?
- 2. How can data science, machine learning, and data visualization help to automate the analysis of large text corpora?
- 3. How can large language models and be used to query documents and translate text?

Informatics and Jurisprudence

What is Informatics?

"Informatics is the study of the behavior and structure of [systems] that generate, store, process and then present information; it is basically **the science of information**. The field takes into consideration the **interaction between information systems and [users]**, as well as the construction of the interfaces between the two [...]."

→ Techopedia

What is Jurisprudence?

"Jurisprudence or legal theory refers to the theoretical study of the propriety of law. Scholars of jurisprudence seek to explain the nature of law in its most general form and provide a deeper **understanding of legal reasoning and analogy, legal systems, legal institutions**, and the proper application and **role of law in society**. [...]"

→ Wikipedia

Informatics and Jurisprudence

Due to the pervasive use of computing technology in modern society, informatics and jurisprudence have many intersections that require computer scientists to have a **basic understanding of law**, and legal scholars to have a **basic understanding of computer science**.

Notes:

Notes:

Example: Algorithms and Decision-Making

exemplary case

In 2011, a German customer receives a **negative score from the credit bureau Schufa**. After requesting an explanation for her negative score, Schufa reveals which data have been used to calculate the score but does not reveal **how the score is calculated**. The customer sues Schufa. In January 2014, the BGH rules that Schufa is not obliged to reveal details of the underlying algorithm.

→ Urt. v. 28.01.2014 Az. VI ZR 156/13

exemplary questions for our course

1. How can we formalize **computational problems** and how can we **use algorithms to solve them**?
2. How can we **implement algorithms** such that they can be executed on a computer?
3. How can machine learning be used to **automate decision-making**?

Press release | 6 July 2022 | Brussels

New rules to improve road safety and enable fully driverless vehicles in the EU

The new [Vehicle General Safety Regulation](#) starts applying today. It introduces a range of mandatory advanced driver assistant systems to improve road safety and establishes the legal framework for the approval of automated and fully driverless vehicles in the EU. The new safety measures will help to better protect passengers, pedestrians and cyclists across the EU, expectedly saving over 25,000 lives and avoid at least 140,000 serious injuries by 2038.

image credit: Website of the European Commission
October 22, 2024

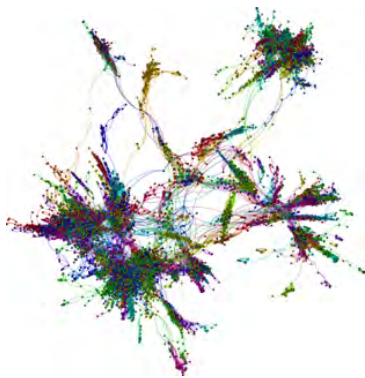
Example: Complexity of Software Systems

exemplary case

In April 2014, a **catastrophic “bug” discovered in the Open Source software OpenSSL** reveals that the majority of online services are vulnerable to attacks that can allow hackers to take control over servers. The **global economic damage** due to stolen data, service downtimes, and additional maintenance efforts is estimated to range in the billions of dollars.

exemplary questions for our course

1. How do we **collaboratively develop complex software systems**?
2. What methods exist to **prevent, find and fix bugs** in software systems?
3. What is **Open Source Software** and how is it developed and maintained?



Graph showing dependencies between components in a complex software system

image credit: own work

Notes:

Notes:

Example: Data Security & Protection

exemplary case

In 2013, the company **Adobe loses usernames and passwords of 150 million customers**. Further analysis reveals that those data were not protected according to accepted industry standards. In 2015, Adobe pays an undisclosed amount to settle customer claims, as well as USD 1.2 Mio in legal fees alone.

exemplary questions for our course

- 1. What techniques can be used to **encrypt information** and how secure are those?
- 2. Which **authentication methods** exist for online services?
- 3. How can communication partners on the Internet **trust** each other?



image credit: theregister.com

Example: Data Storage & Cloud Computing

exemplary case

In 2011, Amazon's cloud service EC2 experiences a major outage, leaving many major online services offline for hours or days. After the service was restored, Amazon acknowledges that some customers experienced a permanent loss of their stored data.

exemplary questions for our course

- 1. How can we **store data** in a computer?
- 2. How can we ensure that data is stored **permanently and consistently**?
- 3. What are **database systems** and how are they used in eCommerce applications?



image credit: www.businessinsider.com

Notes:

Notes:

Example: Internet Communication & Privacy

exemplary case

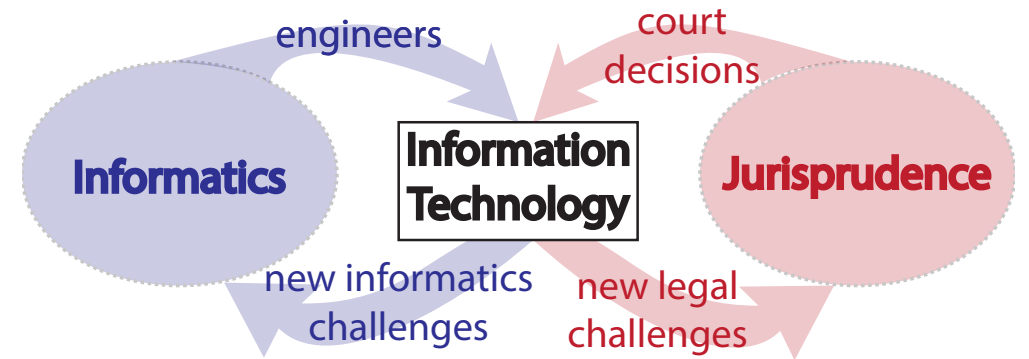
In 2018, an article by Associated Press reports that Google uses its apps to collect location data of smartphone users that have explicitly excluded location data in their privacy settings. In October 2022, Google paid USD 85 mio to settle the case.

exemplary questions for our course

1. How are **Internet technologies** used to communicate globally?
2. How can we use **Web technologies** to implement online services?
3. Which data are generated and stored when **using Web and cloud services**?



Informatics and Jurisprudence



Notes:

Notes:


[illegible]

Dual Course Format

Foundational Concepts → Ingo Scholtes

In the tuesday lectures, we introduce key concepts and methods in informatics.

→ [handout of slides](#)




SPACE-TIME IS LIKE THIS SET OF EQUATIONS, FOR WHICH ANY ANALOGY MUST BE AN APPROXIMATION.

BOOOOOOOING

Practice sessions in python → Anatol Wegner

In the friday practice sessions, we deepen these concepts based on a hands-on python labs

→ [handout of slides](#) + [gitLab repository](#)



WHAT I'M TRYING TO DO IS REALLY SIMPLE.

IT SHOULDN'T BE HARD.

for all students (including those taking 5 ECTS exam **Introduction to Informatics for Jurists**)

for students taking 10 ECTS exam **Introduction to Informatics for Students from all Faculties**

all course material is **available online** via WueCampus

<https://wuecampus.uni-wuerzburg.de/moodle/course/view.php?id=69462>

https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

image credit: Randall Munroe, CC-BY-NC 2.5

Lecture 01: Motivation

Ingo Scholtes

Introduction to Informatics

October 22, 2024

39

I - Digital Data Representation & Computer Architecture

L01 - Motivation

22.10.2024

We give an overview of the history of computing and motivate challenges at the intersection of computer science, law, and humanities. We discuss organizational aspects and give an outline of the course.

L02 - Digital Logics and Data Representation

29.10.2024

We explain basics of digital logics and how we can use it to perform arithmetic operations. We introduce the representation of data in terms of binary numbers.

L03 - Computer Architecture

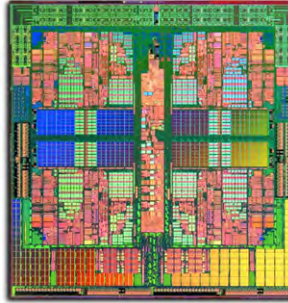
05.11.2024

We introduce the Von Neumann architecture of modern computers and explain how a processor executes computer programs at the machine code level.

L04 - Programming Languages

12.11.2024

We introduce high-level programming languages, explain the difference between compiled and interpreted languages and introduce the popular language python.



microprocessor

II - Algorithmic Thinking and Artificial Intelligence

L05 - Algorithms and Data Structures

19.11.2024

We develop simple algorithms to solve basic computational problems and practice algorithmic thinking. We cover basic data structures used in algorithms.

L06 - Cryptgraphy and Cybersecurity

26.11.2024

We provide an overview of basic encryption and hashing algorithms. We introduce public-key cryptography and explain how it can be used to ensure privacy of communication.

L07 - Introduction to Machine Learning

03.12.2024

We introduce key concepts in artificial intelligence and give examples for unsupervised and supervised machine learning algorithms.

L08 - LLMs and Generative AI

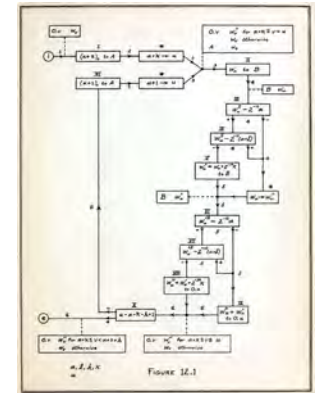
10.12.2024

We introduce deep learning and give an overview of recent AI developments like Large Language Models (LLMs) and generative AI for image and video data.

no lecture

17.12.2024

Due to a trip to the major deep learning conference NeurIPS in Vancouver, there will be no lecture in this week.



flow chart of program, 1947

Notes:

Notes:

III - Data Storage and Communication

L09 - Data Storage

07.01.2025

We explain how data can be stored and accessed in a computer. We explain the memory hierarchy in modern computing hardware and introduce file systems.

L10 - Introduction to Databases

14.01.2025

We show how database technologies can be used to consistently store relational data. We introduce basics of the query language SQL for relational databases.

L11 - Communication Networks

21.01.2025

We cover key concepts in telecommunication and introduce state-of-the-art computer networking techniques like Ethernet or Wifi.

L12 - Internet and WWW

28.01.2025

We introduce the Internet protocol suite and explain how information is routed on the Internet. We cover key distributed Internet service like DNS as well as the WWW protocol http.



I - Foundations of Python Programming

P01 - Getting started

25.10.2024

We introduce the programming environment used throughout the course and explain key concepts of Visual Studio Code, jupyter notebooks, and git.

P02 - Introduction to Python

08.11.2024

We introduce syntax, basic data types and operations of python and show how to execute and debug programs using Visual Studio Code.

P03 - Basic data structures and indexing

15.11.2024

We cover basic python data structures like lists and dictionaries and explain concepts indexing, slicing, and list comprehension.

P04 - Control structures, Loops, and Exceptions

22.11.2024

We show how we can use basic control structures of high-level programming languages to implement simple algorithms.

P05 - Functions and Algorithms

29.11.2024

We show how we can use functions to structure code and how we can use recursion to solve algorithmic problems.



Notes:

Notes:

II - Data Science and AI

P06 - Object-Oriented Programming

06.12.2024

We introduce object-oriented programming and APIs.

P07 - Reading and writing data

13.12.2024

We show how we can use the filesystem to read and write data to permanent storage.

P08 - Basic text and data processing

20.12.2024

We show how we can use python to process large amounts of text and data.

P09 - Classification and Clustering

10.01.2025

We use python and sklearn to implement basic algorithms for classification and clustering.



microprocessor

III - Data Handling and Communication

P10 - SQLite with Python

17.01.2025

We show how we can use SQL to query databases in python.

P11 - Socket communication

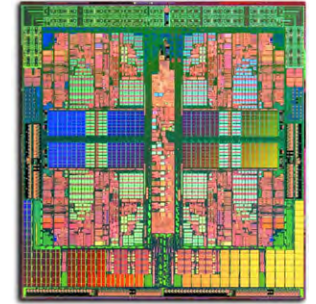
24.01.2025

We introduce communication between processes and machines via sockets.

P12 - HTTP and Web APIs

31.01.2025

We show how we can use HTTP to access services provided in the World Wide Web.



microprocessor

Notes:

Notes:

Exercise Sheets

Group Exercises

- ▶ deepen your knowledge of theoretical concepts
- ▶ group work to develop practical skills
- ▶ gain bonus points for the exam

preliminary schedule

- | | |
|---|----------|
| ▶ assignment 1*: Digital Logics and Data Representation | 29/10/24 |
| ▶ assignment 2: Computer Architecture and Programming | 05/11/24 |
| ▶ assignment 3*: Basic Python Programming | 12/11/24 |
| ▶ assignment 4: Algorithms and Data Structures | 19/11/24 |
| ▶ assignment 5*: Algorithms and Data Structures | 26/11/24 |
| ▶ assignment 6: Machine Learning I | 03/12/24 |
| ▶ assignment 7*: Machine Learning II | 10/12/24 |
| ▶ assignment 8: File Systems and Data Storage | 07/01/24 |
| ▶ assignment 9*: SQL | 14/01/24 |
| ▶ assignment 10: Communication Networks | 21/01/24 |

students of 5 ECTS course **Introduction to Informatics for Jurists** only
need to hand in assignments marked with *



Benjamin Franklin
1706 – 1790

quote

“Tell me and I forget, teach me and I may remember,
involve me and I learn” → Benjamin Franklin

Self-study and exam

final exam

closed-book written examination

- ▶ no programming skills required
- ▶ exercise solutions = bonus points
- ▶ date of final exam: **TUE, February 4 2025, 18:00 - 20:00**

assisting self-study at home

each lecture includes 8 – 10 **self-study questions**

- ▶ repeat most important issues
- ▶ test your comprehension
- ▶ useful to prepare for final exam



image credit: Randall Munroe, CC-BY-SA

Notes:

Notes:

Let's learn from each other!

Humboldtsches Bildungsideal

University education should not be job-focused, but educational training that is independent of economic interests. [...] University should be a place of [...] **exchange between all involved in the scientific process.** → [Wikipedia](#)

- ▶ we need a **pleasant learning environment**
 - ▶ golden rule: there are **no** stupid questions
 - ▶ feel free to **ask anything at any time**

- ▶ contact us in case of problems/concerns

ingo.scholtes@uni-wuerzburg.de
anatol.wegner@uni-wuerzburg.de



Wilhelm von Humboldt
1767 – 1835

Self-study questions

1. What are the advantages of programmable computers like the Z3 compared to earlier mechanical computers or calculators?
2. Name the components of the Von Neumann Architecture and explain the roles of the respective components.
3. What is a Turing machine?
4. Name three technologies that can be used to implement an electronic switch.
5. What are the advantages of a transistor over earlier technologies like vacuum tubes or electromechanical relays?
6. What is a microprocessor?
7. What do the abbreviations CPU and GPU refer to in the context of a modern computer?
8. Define what an algorithm is. Give an example for an algorithm.
9. Give three examples for challenges at the intersection of computer science and law.
10. Give three examples for applications of informatics in the context of digital humanities.

Notes:

Notes:

Literature

reading list

- ▶ C Babbage: **On the Application of Machinery to Computation**, Memoirs of the Astronomical Society of London, 1824
- ▶ A Turing: **On Computable Numbers, with an Application to the Entscheidungsproblem**, Proceedings of the London Mathematical Society, 1937
- ▶ J von Neumann: **First Draft of a Report on the EDVAC**, 1945
- ▶ HH Goldstine, A Goldstine: **The Electronic Numerical Integrator and Computer (ENIAC)**, 1946
- ▶ HH Goldstine, J von Neumann: **Planning and coding of problems for an electronic computing instrument**, Institute for Advanced Study Princeton, 1948
- ▶ A Turing: **Computing, Machinery and Intelligence**, Mind LIX(236), 1950
- ▶ A Hodges: **Alan Turing: The Enigma**, Burnett Books, 1983
- ▶ C Gath: **Governing artificial intelligence: ethical, legal and technical opportunities and challenges**, Philosophical Transactions of the Royal Society A, 2018
- ▶ J Hörnle: **Internet Jurisdiction Law and Practice**, Oxford University Press, 2021



Notes:

Introduction to Informatics for Students from all Faculties

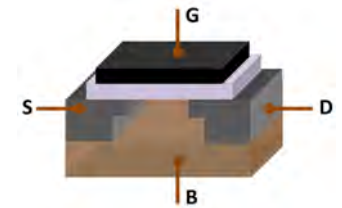
Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

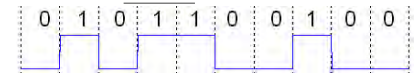
Motivation

- ▶ how are data and instructions represented in an electronic digital computer?
- ▶ how can we implement logical operations based on digital electronic circuits?
- ▶ how can we perform (arithmetic) operations on digitally encoded data?
- ▶ how can we **represent instructions** that can be executed by a programmable computer? → L03
- ▶ answering these questions requires basic foundations in terms of **numeral systems** and **digital logics**



schematic view of a transistor

image credit: Wikipedia, CC-BY-SA



a digital signal

image credit: Wikipedia, CC-BY-SA

Notes:

- **Lecture L02: Digital Logics and Data Representation** 29.10.2024
- **Educational objective:** We show how data is represented in a digital computer. We introduce basics of digital logics and show how we can implement arithmetic operations based on logic circuits.
 - Digital Representation of Data
 - Digital Logics and Digital Circuits
 - From Logics to Arithmetics
- **Exercise Sheet 01** due 05.11.2024

Notes:

Encoding numbers with numeral systems

definition

A **numeral system** is a system that can be used to consistently encode numbers based on a set of symbols that are called **digits**. In a **positional numeral system** the contribution of a digit to the value of the encoded number depends on the **position of the digit**.

example: decimal numeral system

- ▶ digits represent different **powers of ten** depending on their position
- ▶ we call ten **base** of the decimal number system
- ▶ rightmost position represents **power of zero**, i.e. $10^0 = 1$
- ▶ powers of ten associated with a position increase from right to left

- ▶ value of encoded number is given by the **sum of contributions of individual digits**
- ▶ **left-most digit** is called **most-significant digit**
- ▶ **right-most digit** is called **least-significant digit**

MMXXIV

number 2024 represented in the **Roman numeral system**, where M represents thousand, X represents ten, V represents five and I represents one

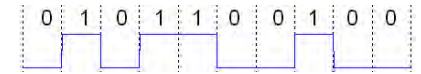
$$\begin{array}{cccc} 10^3 & 10^2 & 10^1 & 10^0 \\ 2 & 0 & 2 & 4 \end{array}$$

$$2 \cdot 1000 + 2 \cdot 10 + 4 \cdot 1 = 1011$$

number 2024 represented in the **decimal numeral system**, where the value of a digit depends on its position

Binary numeral system

- ▶ in a digital electronic computer we can use voltage levels to encode two symbols 0 and 1
 - ▶ low or no voltage = 0
 - ▶ high voltage = 1



- ▶ voltage levels can be used to encode binary digits or “bits” (0 or 1) of the **binary numeral system**

- ▶ positional encoding analogous to decimal system, but using **base two instead of ten**

- ▶ depending on their position **digits represent powers of two**, where the least-significant bit represents $2^0 = 1$

- ▶ value of an encoded number is given as the **sum of powers of two**

$$\begin{array}{cccc} 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 1 & 1 \end{array}$$

$$1 \cdot 8 + 1 \cdot 2 + 1 \cdot 1 = 11$$

image credit: Wikipedia, CC-BY-SA

Notes:

Notes:

Powers of two

2^1	2	2^9	512	2^{17}	131,072	2^{25}	33,554,432
2^2	4	2^{10}	1,024	2^{18}	262,144	2^{26}	67,108,864
2^3	8	2^{11}	2,048	2^{19}	524,288	2^{27}	134,217,728
2^4	16	2^{12}	4,096	2^{20}	1,048,576	2^{28}	268,435,456
2^5	32	2^{13}	8,192	2^{21}	2,097,152	2^{29}	536,870,912
2^5	64	2^{14}	16,384	2^{22}	4,194,304	2^{30}	1,073,741,824
2^7	128	2^{15}	32,768	2^{23}	8,388,608	2^{31}	2,147,483,648
2^8	256	2^{16}	65,536	2^{24}	16,777,216	2^{32}	4,294,967,296

rules of thumb for orders of magnitude

$2^0 = 1 = 10^0$
 $2^{10} \approx 1,000 = 10^3$
 $2^{20} \approx 1,000,000 = 10^6$
 $2^{30} \approx 1,000,000,000 = 10^9$
 $2^{40} \approx 1,000,000,000,000 = 10^{12}$

...

Group Exercise 02-01

► Convert the the decimal number 126 into the binary number system.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	We thus have
128	64	32	16	8	4	2	1	
0	1	1	1	1	1	1	0	

$126 = 64 + 32 + 16 + 8 + 4 + 2$

► Convert the binary number 10010101 into the decimal number system.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	And thus $128 + 16 + 4 + 1 = 149$
1	0	0	1	0	1	0	1	

Hexadecimal numbers

- ▶ long binary numbers are **difficult to read and memorize for humans**
- ▶ **hexadecimal numeral system** (base 16) yields human-friendly representation of large (binary) numbers
- ▶ to distinguish 16 numbers from 0 to 15 with single digit, we extend the symbols 0, . . . , 9 by letters A, . . . F
- ▶ **prefix 0x** used to denote hexadecimal number, e.g. 0x10 = 16
- ▶ since $2^4 = 16$, each hexadecimal digit corresponds to **four bits**, i.e. **easy to convert binary/hexadecimal numbers**

0101101011110011

	16 ³	16 ²	16 ¹	16 ⁰
0x	5	A	F	3

$$5 \cdot 4096 + 10 \cdot 256 + 15 \cdot 16 + 3 \cdot 1 = 23283$$

0x	5	A	F	3
	0101	1010	1111	0011

Group Exercise 02-01

1/2

Convert the following decimal numbers into the **hexadecimal and binary numeral system**.

- ▶ 3

3 = 0x3 = 0011
- ▶ 4

4 = 0x4 = 0100
- ▶ 19

19 = 0x13 = 0001 0011
- ▶ 64

64 = 0x40 = 0100 0000
- ▶ 255

255 = 0xFF = 1111 1111

Notes:

Notes:

Group Exercise 02-02

2/2

Convert the following hexadecimal numbers into the **decimal and the binary numeral system**.

- 0x10

0x10 = 16 = 0001 0000
- 0xF0

0xF0 = 240 = 1111 0000
- 0xAA

0xAA = 170 = 1010 1010
- 0x0100

0x100 = 256 = 0001 0000 0000
- 0xFFFF

0xFFFF = 65535 = 1111 1111 1111 1111

Encoding text

- how can we **encode text in a digital computer**?
- idea: use numbers to encode text characters and represent each number by group of bits

ASCII encoding

American Standard Code for Information Interchange (ASCII) defines a 7-bit encoding of 128 different characters.

- code table maps numbers 0 to 127 (represented by 7 bits) to characters and vice-versa

example

Binary ASCII-encoded text "Jurist"
1001010 1110101 1110010 1101001 1110011 1110100
Hexadecimal ASCII-encoded text "Jurist"
0x 4A 75 72 59 73 74

- UNICODE text encoding supports **all writing systems** in the world

USASCII code chart

							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7
							0	1	2	3	4	5	6	7

ASCII code table from printer handbook, 1971

image credit: public domain

Notes:

Notes:

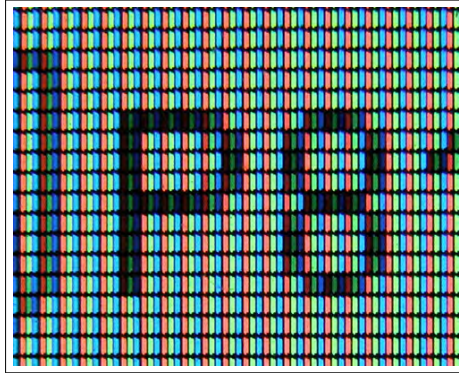
Encoding images

- ▶ how can we represent **image data** in a digital computer?

digital images

A digital (raster) image is a picture that is composed of a **rectangular arrangement of pixels**, where each pixel either represents a brightness and (possibly a color value).

- ▶ **idea**: use numbers to represent brightness (and colors) of pixels
 - ▶ **grayscale pixels**: 8 bits encoding 255 brightness levels from black (0) to white (255)
 - ▶ **color pixels**: 3×8 bits encoding 255 brightness levels of red (R), green (G), blue (B)
- ▶ image can be digitally encoded by **sequence of bits**, where groups of 8 or 24 bits represent grayscale or color pixels in rectangular grid

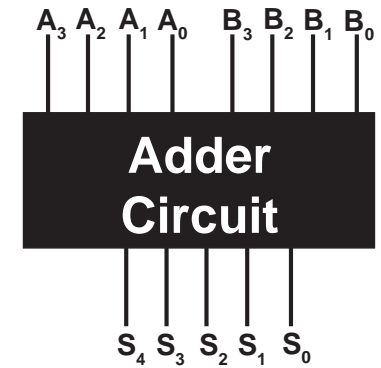


Closeup of pixels (consisting of a red, green, and blue subpixel) on a liquid crystal display (LCD) laptop screen

image credit: Wikimedia Commons, User Kprateek88, CC-BY-SA 4.0

From binary numbers to arithmetics

- ▶ we introduced the **digital representation of numbers, text, and images** by bits
- ▶ how can a digital computer perform **arithmetic operations** like addition, multiplication, etc.?
- ▶ as example, consider **addition of two binary numbers with 4 bits each**
- ▶ given **input of 8 bits** (i.e. binary numbers A and B with 4 bits each), we must compute **5 bits of output** that encode the sum $A + B$



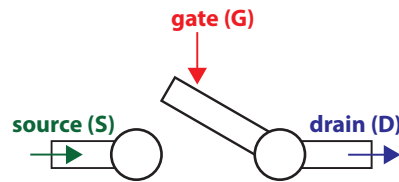
Notes:

Notes:

- Why do we need five bits for the sum of two four bit binary numbers?
- The largest sum that we can have for two four bit binary numbers is the sum of $1111_b = 15$ and $1111_b = 15$ which is $30 = 1110_b$. This requires 5 bits.

Digital logics

- ▶ **digital logics** is the basis for any (arithmetic) operation of a digital computer
- ▶ all functions of a digital computer can eventually be reduced to simple logical operations implemented by electronic switches (e.g. transistors)
- ▶ consider one bit representing voltage levels of source (S), gate (G), and drain (D)
- ▶ considering S and G as inputs, the output D represents a **logical AND operation**



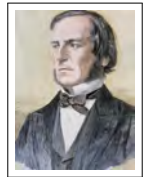
S	G	D
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Logic

- ▶ formal treatment of logics pioneered by mathematician **George Boole**
- ▶ Boole **formalized logics** in analogy to arithmetic operations like $+$, $-$, \times → G Boole, 1854
- ▶ basic logical operations AND, OR and NOT can be mapped to **arithmetic operations** on numbers 0 and 1 representing False and True

basic operations in Boolean logics

- ▶ $A \text{ AND } B \equiv A \times B$
- ▶ $A \text{ OR } B \equiv A + B - A \times B$
- ▶ $\text{NOT } A \equiv 1 - A$



George Boole (1815 – 1864)

image credit: Wikimedia Commons, public domain

Notes:

Notes:

Truth tables

- ▶ **truth tables** define output of logical operations
- ▶ each row in the truth table is one **possible combination of inputs**
- ▶ we use 0 and 1 to represent logical values False and True

AND		
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR		
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
A	NOT A
0	1
1	0

XOR		
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

basic operations in Boolean logics

- ▶ $A \text{ AND } B \equiv A \times B$
- ▶ $A \text{ OR } B \equiv A + B - A \times B$
- ▶ $\text{NOT } A \equiv 1 - A$
- ▶ $A \text{ XOR } B \equiv (A - B)^2$

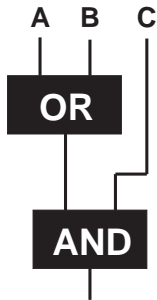
Boolean functions and digital circuits

- ▶ basic Boolean logical operations NOT, AND, OR and XOR are defined for one or two inputs, respectively
- ▶ we can use these as building blocks of more complex **Boolean functions** with more than one or two inputs
- ▶ in formular notation we use brackets to **determine order in which operations are executed**

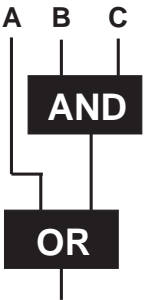
examples

- ▶ $A \text{ OR NOT } B$
- ▶ $[A \text{ OR } B] \text{ AND } C$
- ▶ $A \text{ OR } [B \text{ AND } C]$

- ▶ Boolean functions can be represented as **digital circuits** where **logic gates** represent Boolean operations



digital circuit
implementing Boolean
function
 $[A \text{ OR } B] \text{ AND } C$



digital circuit
implementing Boolean
function
 $A \text{ OR } [B \text{ AND } C]$

image credit:

Notes:

Notes:

Group Exercise 02-03

1/4

► Give the truth table for the Boolean function $A \text{ OR } [B \text{ AND NOT } C]$ with three inputs A, B, C .

A	B	C	$A \text{ OR } [B \text{ AND NOT } C]$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Group Exercise 02-03

2/4

► Give a formula for the Boolean function represented by the following truth table.

A	B	C	?
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

This can be expressed as $[B \text{ XOR } C] \text{ OR } [A \text{ AND } B \text{ AND } C]$

Notes:

Notes:

Group Exercise 02-03

3/4

► Show that the **logical operation XOR** (with two inputs *A* and *B*) can be constructed as Boolean function that only uses AND, OR and NOT operations.

A	B	A OR B	NOT [A AND B]	[A OR B] AND NOT [A AND B] = A XOR B
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Notes:

Group Exercise 02-03

4/4

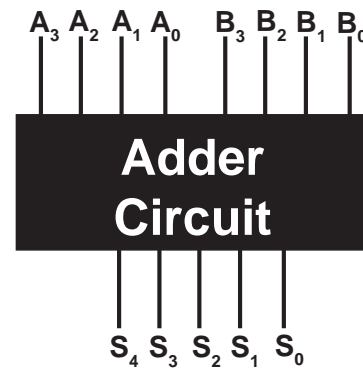
► Show that the **logical operation OR** (with two inputs *A* and *B*) can be constructed as Boolean function that only uses AND and NOT operations.

A	B	NOT A	NOT B	NOT [[NOT A] AND [NOT B]] = A OR B
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

Notes:

From Digital Circuits to Arithmetics

- ▶ we can implement complex Boolean functions by composing basic logical operations AND, OR, and NOT
- ▶ assume that we have digital inputs that represent **two numbers A and B in the binary numeral system**
- ▶ how can we perform **arithmetic operations** like addition or multiplication?
- ▶ idea: specify **Boolean functions** that give correct digital output for each combination of digital inputs



Adding decimal numbers

- ▶ consider **pencil-and-paper algorithm** to **add numbers** in the decimal numeral system

$$\begin{array}{r} 1 \quad 2 \quad 5 \quad 7 \\ 2_1 \quad 9 \quad 3 \quad 2 \\ \hline 4 \quad 1 \quad 8 \quad 9 \end{array}$$

pencil-and-paper algorithm to add two numbers

- step 1 start at right-most position
- step 2 add digits at current position
- step 3 write last digit of sum below current position
- step 4 for sums ≥ 10 additionally **carry over 1** to position on the left
- step 5 move one position to left and go to step 2

- ▶ algorithm reduces addition of numbers with **any number of digits** to repeated addition of **single digit numbers**

open questions

- ▶ how can we apply this to binary numbers ?
- ▶ how can we map addition to logical operations?

Notes:

Notes:

Adding binary numbers

- ▶ how does pencil-and-paper addition work for two **binary numbers** with arbitrary number of digits?
- ▶ we can apply the same algorithm but we only have binary digits (bits) 0 and 1

$$\begin{array}{r} 0 \quad 1 \quad 1 \quad 0 \\ 0_1 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 1 \quad 1 \end{array}$$

pencil-and-paper algorithm to add two numbers

- step 1 start at right-most position
- step 2 add digits at current position
- step 3 write last digit of sum below current position
- step 4 for sums ≥ 2 additionally **carry over 1** to position on the left
- step 5 move one position to left and go to step 2

- ▶ **carry bit** of one is created whenever the sum is larger than base two of the binary numeral system

Half adder circuit

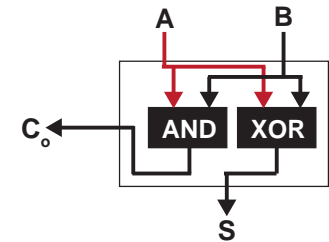
- ▶ we can write down the truth table of a Boolean function that generates the **sum** S for two binary digits A and B
- ▶ sum bit can be generated by a **single XOR operation** on A and B
- ▶ we can further write a Boolean function that generates the **carry-over bit** C_o
- ▶ carry bit can be generated by a **single AND operation** on A and B
- ▶ we call the resulting digital circuit a **half adder**
- ▶ is this enough to add two binary numbers with **any number of digits**?

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

$$S = A \text{ XOR } B$$

A	B	C_o
0	0	0
0	1	0
1	0	0
1	1	1

$$C_o = A \text{ AND } B$$

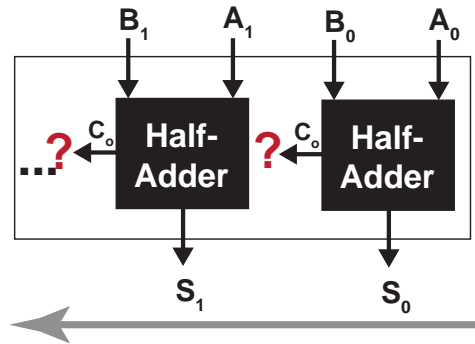


Notes:

Notes:

Adding numbers with more than one bit?

Can we connect multiple half adder circuits to add numbers A and B that consist of more than one bit?



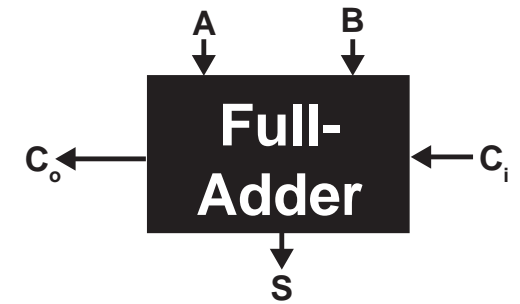
problem

Half adder produces carry-out bit for next position but does not have input that allows to consider carry bit from previous position!

Full adder circuit

- ▶ paper-and-pencil algorithm requires to **include carry bit** from the right when summing digits for any given position
- ▶ need Boolean function taking three bits A , B and “**carry-in**” bit C_i as input and generating sum S and “**carry-out**” bit C_o as output
- ▶ resulting digital circuit is called **full adder**
- ▶ **carry-out output** generated for one position is used as **carry-in input** for next position to the left
- ▶ this **digital circuit design** implements our simple pen-and-pencil method

0	1	1	0
0 ₁	1 ₀	0 ₀	1 ₀
1	0	1	1



Notes:

Notes:

Group Exercise 02-04

- ▶ Write down truth tables for the sum and carry-out bit of a full-adder.
- ▶ Give formulas for Boolean functions for the sum and the carry-out bit of a full-adder.

A	B	C_i	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$S = A \text{ XOR } B \text{ XOR } C_i$

A	B	C_i	C_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

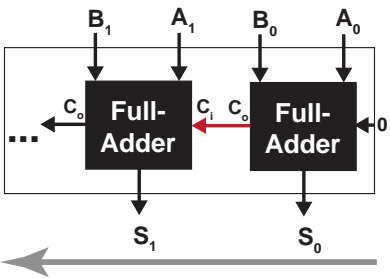
$C_o = [C_i \text{ AND } [A \text{ XOR } B]] \text{ OR } [A \text{ AND } B]$

Carry-Ripple Adder

- ▶ we can now construct a **full adder** based on logical operations AND, OR and XOR.
- ▶ we connect multiple full adders to a digital circuit that adds binary numbers with **any number of digits**

idea for adding two 4 bit numbers

- ▶ use sequence of 4 full adders, one for each pair of digits of inputs A and B
- ▶ each full adder generates one bit of sum S
- ▶ connect carry-out of each full adder with carry-in of next full adder in sequence
- ▶ last carry-out is **most-significant bit** of the sum
- ▶ we can set **carry-in of first full adder** to zero
- ▶ we call this design a **carry-ripple adder**

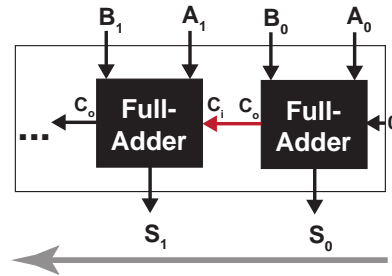


Notes:

Notes:

Computation time of Carry-Ripple Adder

- ▶ what is the **computation time** of a carry-ripple adder?
- ▶ after setting input bits, we must wait until all carry bits have “rippled” through sequence of full adders
- ▶ “gate delay” until sum is computed corresponds to number of full adders in sequence
- ▶ assume that it takes full adder **one nanosecond** ($= 10^{-9}$ = 1 billionth of a second) to compute sum of two bits
- ▶ adding two 32 bit numbers with carry-ripple adder would then take **32 nanoseconds**, i.e. approx. 31 million additions per second
- ▶ in practice we use other designs like **carry-lookahead** or **carry-select adder** that speed up computation



Conclusion

- ▶ we have shown how we can **digitally represent data like numbers, text, or images**
- ▶ we introduced **basic operations in Boolean logics** like AND, OR, NOT, and XOR
- ▶ we expressed **Boolean functions** given in a truth table using basic Boolean operations
- ▶ we demonstrated how to implement arithmetic operations on binary representations of numbers using **digital circuits**
- ▶ digital logics is the **foundation of all digital computers and technology**

USASCII code chart

		Character							
		0	1	2	3	4	5	6	7
0	0	NUL	DLE	SP	0	@	P	\	p
0	1	SOH	DC1	!	1	A	Q	a	q
0	2	STX	DC2	"	2	B	H	b	r
0	3	ETX	DC3	#	3	C	S	c	s
0	4	EDT	DC4	\$	4	D	T	d	t
0	5	ENQ	NAK	%	5	E	U	e	u
0	6	ACK	SYN	&	6	F	V	f	v
0	7	BEL	ETB	'	7	G	W	g	w
0	8	BS	CAN	(8	H	X	h	x
0	9	HT	EM)	9	I	Y	i	y
0	10	LF	SUB	*	10	J	Z	j	z
0	11	VT	ESC	+	11	K	[k	{
0	12	FF	FS	,	12	L	\	l	
0	13	CR	GS	-	13	M]	m	}
0	14	SO	RS	.	14	N	^	n	~
0	15	SI	US	/	15	O	_	o	DEL

ASCII code table

image credit: public domain

Notes:

Notes:

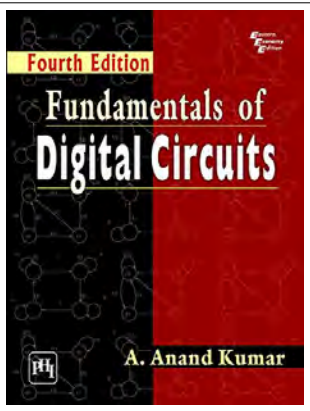
Self-study questions

- 1. What is a bit?
- 2. Give an example for a positional and a non-positional numeral system to represent numbers.
- 3. Convert the decimal number 42 into the binary numeral system.
- 4. Convert the binary number 101010 into the decimal numeral system.
- 5. Convert the hexadecimal number 0x2A into the decimal and the binary numeral system.
- 6. Explain how you can use electronic switches to implement the basic logical operations AND, OR, and NOT.
- 7. Use a truth table to explain the difference between the OR and the XOR operation.
- 8. Give the truth table for the Boolean formula $[A \text{ OR } \text{NOT } B] \text{ AND } C$.
- 9. Use a truth table to show that the logical operator $X \text{ OR } Y$ corresponds to NOT ($\text{NOT } X \text{ AND } \text{NOT } Y$).
- 10. Give the truth table for the outputs of a half and a full adder.
- 11. Explain the difference between a half and a full adder.
- 12. What is the largest output that a carry-select adder for two four-bit inputs and a carry-in input can produce?
- 13. Draw a diagram of the digital circuit implementation of a full adder.

Literature


reading list


- ▶ A Anand Kumar: **Fundamentals of Digital Circuits**, PHI Learning, 2016
- ▶ K Fricke: **Digitaltechnik**, Springer Vieweg, 2018
- ▶ U Schöningh: **Logik für Informatiker**, Spektrum, 2005
- ▶ C Meinel, M Mundhenk: **Mathematische Grundlagen der Informatik**, BG Teubner, 2000



Notes:


Notes:





Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes
 Chair of Machine Learning for Complex Networks
 Center for Artificial Intelligence and Data Science (CAIDAS)
 Julius-Maximilians-Universität Würzburg
 Würzburg, Germany
ingo.scholtes@uni-wuerzburg.de



Lecture 03
Computer Architecture and HW/SW Interface
 November 5, 2024

Motivation

- ▶ we reviewed milestones in the **history of computing**
- ▶ we explained how numbers, text, and image data are digitally represented in a computer
- ▶ we introduced **Boolean logics** and used it to implement a **digital circuit performing an arithmetic operation**

open questions

- ▶ what is the **architecture of programmable computers**
 - ▶ how are **instructions digitally stored** in a computer?
 - ▶ how are **programs executed** by a modern computer?
- ▶ we will introduce the **Von Neumann architecture of stored-program computers**
 - ▶ we adopt a **top-down approach**, incrementally adding technical details up to the bit level



John von Neumann, 1903 – 1957

image credit: Los Alamos National Security, LLC

Notes:

- **Lecture L03: Computer Architecture and HW/SW Interface** 05.11.2024
- **Educational objective:** We study how computer programs are executed at the level of machine instructions, investigate technical details of the central processing unit of modern computers and explain the key functionality of operating systems.
 - Von Neumann Architecture
 - Architecture of a CPU
 - Hardware-Software Interface
 - Operating Systems
- **Exercise sheet 2** due 12.11.2024

Notes:

Architecture of modern computers



Macbook computer

image credit: Wikipedia, VladIsLoveW, CC-BY-SA

which components can we see?

▶ input devices

- ▶ keyboard
- ▶ touchpad
- ▶ camera
- ▶ USB ports

▶ output devices

- ▶ monitor
- ▶ speakers
- ▶ USB ports

Looking inside a computer ...



mainboard (or motherboard) of a workstation computer

image credit: Wikipedia, User leibnizkeks, CC-BY-SA

which components can we see?

- ▶ **central processing unit** (CPU) (under cooling fan)
- ▶ **memory** modules (RAM)
- ▶ interfaces for **input/output devices**

Notes:

Notes:

Von Neumann architecture

- ▶ von Neumann introduced concept of **stored-program computer**, which stores both data and instructions (i.e. the “program”) in memory

Von Neumann architecture → J von Neumann, 1945

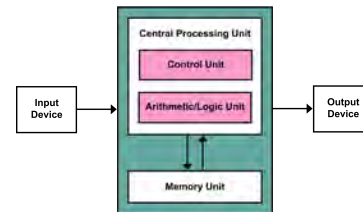
1. **input and output (I/O)**
2. **memory unit**, which stores both data and instructions
3. **arithmetic and logical unit (ALU)**, which implements arithmetic operations like addition, subtraction, multiplication as well as number comparisons or logical operations
4. **control unit (CU)**, which decodes instructions and orchestrates their execution

- ▶ Von Neumann architecture is basis for (almost) all modern computers

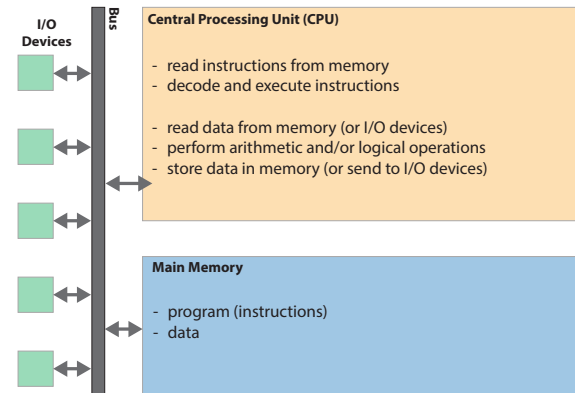


John von Neumann, 1903 – 1957

image credit: Los Alamos National Security, LLC



Central processing unit (CPU)



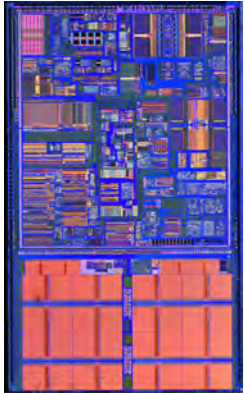
definition

The Central Processing Unit (CPU) executes instructions that are stored in memory. These instructions can perform arithmetic or logical operations on data stored in main memory. A “databus” (or short “bus”) is used to connect the CPU to memory and I/O devices.

Notes:

Notes:

What's inside the CPU?



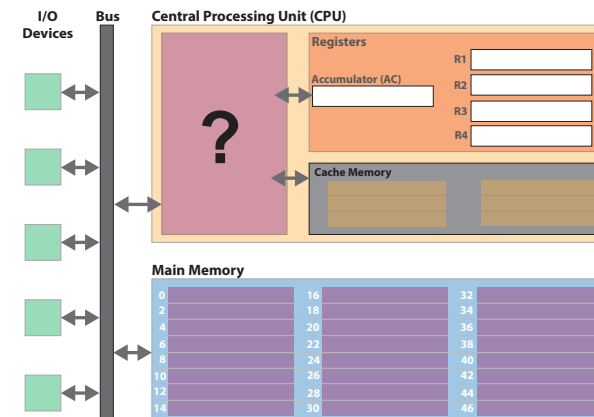
Intel Pentium II Dixon (CPU)
27 mio transistors, ca. 1997

image credit: Wikipedia, Public Domain

which components can you see?

- ▶ **cache memory** (bottom part)
 - ▶ fast and expensive memory to speed up (slow) access to main memory (RAM) via the databus
 - ▶ typically small
 - ▶ few million bytes (MB) cache vs. few billion bytes (GB) RAM
→ L09 - Data Storage
- ▶ **registers** (regular structures in top part)
 - ▶ extremely fast storage location for individual numbers/values
 - ▶ typically 32 or 64 bits per register
 - ▶ a few dozen registers in total
 - ▶ CPU uses registers to store intermediate values during execution of instruction
- ▶ many different **functional units** (components in top part)

Main memory, cache, and registers



Notes:

- If we would unmount the large cooling fan on the mainboard shown before, detach a metal heat exchanger (the so-called heat sink) and then finally open the housing of the CPU underneath, we might see a structure like the one shown above.

Notes:

- Let us now add some details to the architectural view of a modern computer. Inside the CPU we have multiple registers. Typically, those are just numbered (e.g. R1 - R32).
- Each CPU also has at least one special register that is called the Accumulator (AC), which holds the result of the current operation. As an example, we could perform an addition of two numbers stored in the registers R1 and R2 and the result $R1 + R2$ would then be stored in the Accumulator.
- In the overview above, we also added some details on the main memory and the cache memory. In both, we can access individual values (typically at the level of a single byte = 8 bits). This can be done by enumerating individual bytes and assign each a number, which is used as a so-called **memory address**.
- In addition, there is also a so-called **word size**, which can contain multiple bytes and which typically corresponds to the number of bits that are used to digitally represent memory addresses, instructions, and numbers in a given computer architecture.
- In the example above, each word in the main and cache memory consists of two individually addressable bytes (8 bit each), i.e. the word width of this computer would be $2 \cdot 8 = 16$ bits. This would allow us to perform arithmetic operations on integer numbers from 0 to $2^{16} - 1 = 65535$ (→ L02 - Digital Representation of Data) and in each word we can store a 16-bit memory address that allows to address 65536 different bytes from address 0 to address 65535. 16-bit computers, which support a main memory of 64 Kilobytes were common in the 1960s and 1970s. Real modern computers typically have a word width of 64 bits.

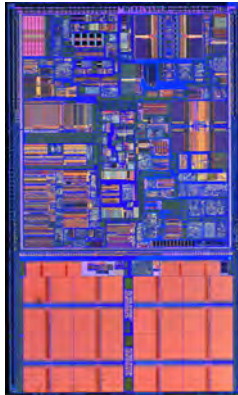
How can we “program” a computer?

- ▶ we can change the “program” of a **stored-program computer**
- ▶ program = sequence of simple **machine instructions** that are executed by the central processing unit (CPU)

examples for machine instructions

- ▶ arithmetic operation on numbers in registers (e.g. add, multiply, etc.)
- ▶ comparison of numbers in registers (e.g. greater, smaller, equal)
- ▶ logical operations on numbers in registers
- ▶ load/store register values to/from memory
- ▶ change next instruction to be executed

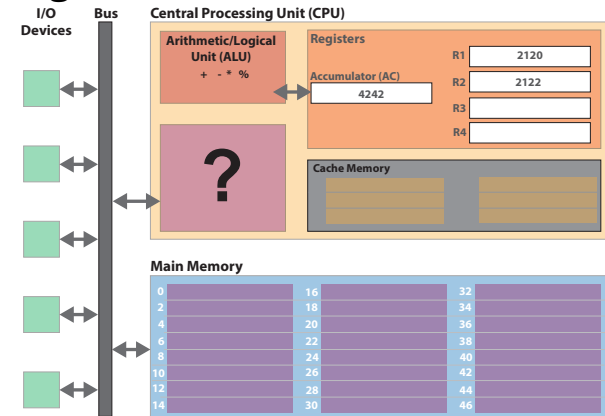
- ▶ need one component that performs **arithmetic and logical operations** and one that **orchestrates execution of instructions in memory**



Intel Pentium II Dixon (CPU)
27 mio transistors, ca. 1997

image credit: Wikipedia, Public Domain

Arithmetic-Logical Unit (ALU)



definition

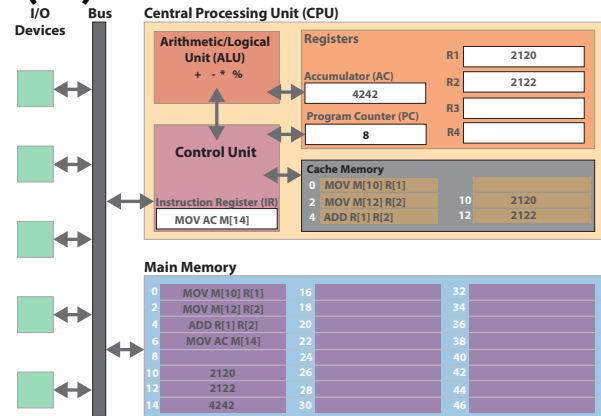
The **Arithmetic/Logical Unit (ALU)** performs a given arithmetic and logical operation on inputs stored in registers and stores the result in the accumulator (AC) register.

Notes:

Notes:

- The ALU consists of digital circuits implementing those arithmetic operations (such as the Carry-Ripple Adder introduced last week). In fact, it is enough to implement addition as arithmetic operation, since other arithmetic operations like subtraction, multiplication, and division can be reduced to multiple addition operations.
- To make arithmetics fast, the ALU in modern CPUs can perform a number of complex arithmetic operations in a single step.

Control Unit (CU)



definition

The **Control Unit** loads the next instruction from memory into the Instruction Register (IR), decodes it, loads operands into registers, directs the operation to be performed by the **Arithmetic/Logical Unit (ALU)**, and updates the Program Counter (PC), which holds the address of the next instruction to be executed.

Digitally encoding instructions

- CPU executes programs given as instructions in **low-level machine code**
- **instruction and arguments** are encoded as bit patterns
- instructions and their meaning is defined by **processor architecture** (e.g. x86, x64, ARM)

exemplary instructions of Intel x86 processors

Op-Code	Mnemonic	Meaning
0x00	ADD	Add two registers
0x48	DEC	Decrement register by one
0x83	SUB	Subtract two registers
0x89	MOV	Move to/from registers
0x8d	LEA	Load effective address
0xe8	CALL	Call procedure

- machine code is **specific to processor (architecture)**

toy examples (not real x86 instructions!)

// load value from memory address 0xA and store it in register 2

Opcode	Operand 1	Operand 2
1000 1001	1010	0010
0x89	0xA	0x2

// load value from memory address 0xC and store it in register 4

Opcode	Operand 1	Operand 2
1000 1001	1100	0100
0x89	0xC	0x4

// Add numbers in registers 2 and 4 and store result in accumulator

Opcode	Operand 1	Operand 2
0000 0000	0010	0100
0x00	0x2	0x4

// store result at memory address 0xE

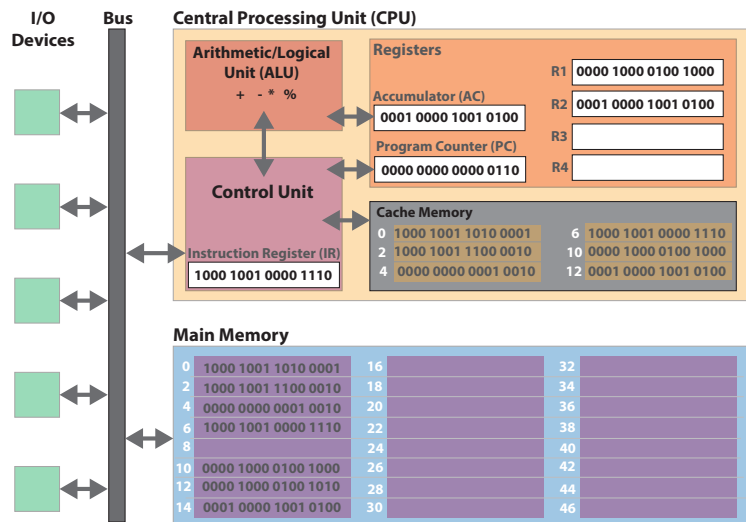
Opcode	Operand 1	Operand 2
1000 1001	0000	1110
0x89	0x0	0xE

Notes:

- In the slide transitions above, we give a detailed step-by-step visualization of how the program stored as machine instructions in the memory addresses 0 — 7 in main memory would be executed on two 16-bit input numbers, stored at the memory addresses 10 — 13.
- In this exemplary execution, we also show how the cache memory works. Whenever we access a memory address, the CPU first checks whether this address is already stored in the cache. If not, one of the existing values in the cache is replaced and the new value is stored, so it can be accessed more quickly next time.

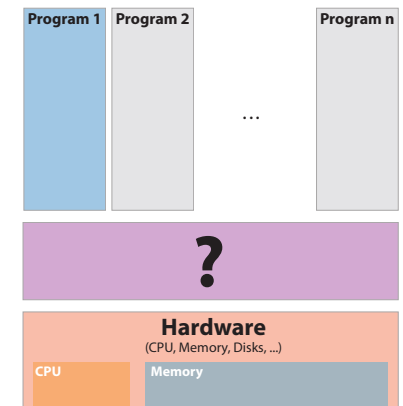
Notes:

The Hardware/Software Interface



Are we done yet?

- ▶ we now have a **basic understanding how a computer executes a program**, which is a sequence of digitally encoded machine instructions
- ▶ in principle, we could set bit patterns that encode instructions and data in memory, start computer, and wait until the CPU finishes the execution



open questions

- ▶ how can we interact with computers in a human-friendly way?
- ▶ how can we run and manage programs in practice?
- ▶ how can we run multiple programs in parallel?
- ▶ how can those programs access I/O devices?

- ▶ idea: use **intermediary software** that helps us to operate programmable computers

Notes:

- We now know how both data and instructions can be digitally encoded in terms of bit patterns. In the last lecture, we have seen how arithmetic operations on such bit patterns (encoding numbers) can be implemented in terms of digital circuits. This is the basis to implement the operations provided by an arithmetic-logical unit (ALU). An additional so-called multiplexer circuit then allows us to select which operation should be performed on the inputs (see self-study questions).
- By writing down Boolean functions and implementing them in terms of digital circuits, we can implement the Control Unit (CU) of the processor in a very similar fashion.

Notes:

What is an Operating System?



Windows 11 Desktop

definition

"An **operating system (OS)** is a **system software** that manages computer hardware, software resources, and provides common services for computer programs. [...] It acts as **intermediary** between programs and computer hardware." → [Wikipedia](#)

key functionality

- ▶ allows to **launch and execute multiple programs in parallel** (multi-tasking)
- ▶ provides **common services** for programs like, e.g., unified graphical user interface, reading/writing of data, network communication, printing, etc.
- ▶ provides **access to hardware** like CPU, memory, IO devices

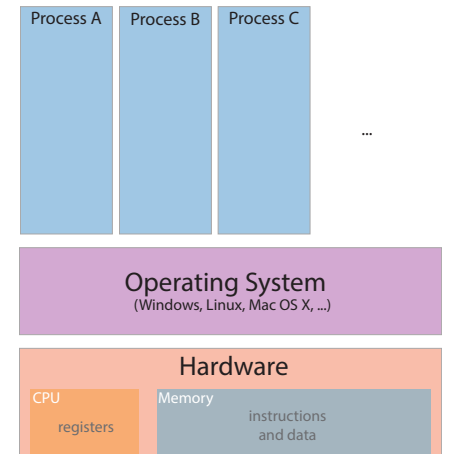
OS Processes

- ▶ **process** is an instance of a computer program executed by the CPU

definition

A **process** is an instance of a computer program executed by the CPU. The current state of a process is fully determined by the **instructions and data in memory**, and its current execution status (i.e. the **contents of CPU registers**).

- ▶ processes are **launched and managed by OS**
- ▶ **multiple processes** can be launched that execute the **same instructions** in memory
- ▶ for security, OS ensures that processes are **isolated from each other**
- ▶ we can use OS tools like TaskManager or ps to **list and kill running processes**



Notes:

Notes:

- Importantly, the OS must ensure that different processes are completely isolated from each other, i.e. one process can not access the data or instructions of another running process. Two processes can only communicate with each other explicitly (e.g. by sharing data via files or databases, or sending messages to each other).
- This isolation is crucial for reasons of security, i.e. a process running on your computer should not be able to read the memory of the process running your online banking application.
- The isolation of processes is also crucial in order to avoid an accidental corruption of memory, i.e. if there was no isolation a bug in one process could actually crash other processes as well. This was an issue in very early operating systems that did not offer this isolation.
- It would take too far to discuss this in detail, but the isolation between processes is guaranteed based on the provision of a **virtual addresses space** for each process. I.e. each process can access any memory address in a "virtual address space", which the OS maps to a physical address in memory. This allows the OS to maintain different mappings from virtual to physical memory addresses, which prevents one process to access memory of another process.

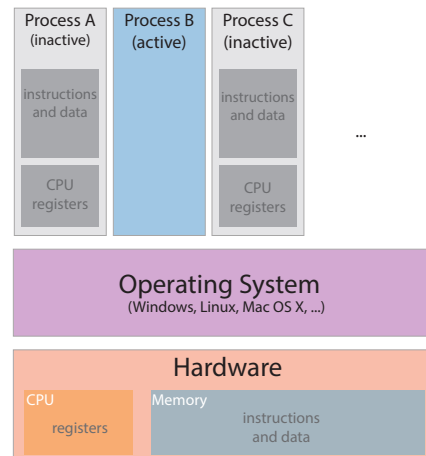
Multi-Tasking

- ▶ OS can use **multi-tasking** to execute **multiple processes concurrently** (even on a single CPU)
- ▶ every few milliseconds, OS performs **context switch** between running processes
- ▶ context switch from process *A* to *B* requires to **switch execution context**

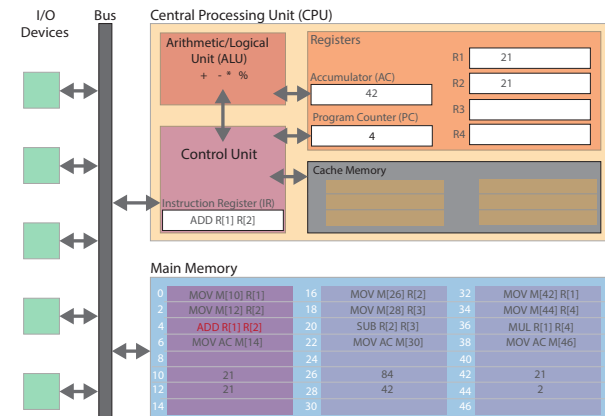
context switch from process *A* to *B*

1. interrupt execution of program by CPU
2. save current values in CPU registers (incl. PC) to memory, which fully determine execution state of process *A*
3. restore previously saved CPU registers of process *B* from memory
4. continue execution of program by CPU

- ▶ **OS scheduler** fairly allocates CPU time
- ▶ **preemptive scheduler** forces context switches



Context switches



context switches

A context switch requires to **save/restore CPU registers that hold the execution state of a process**. Due to the need to access main memory, this is a costly operation. It can further lead to a **“cold cache”** that is either empty or populated with irrelevant values.

Notes:

1. The opposite of preemptive scheduling is called cooperative scheduling. This means that a context switch can only happen if a process “voluntarily” surrenders the CPU periodically, such that another process can take over. Early operating systems like Windows (before Windows 95) or Mac OS (before Mac OS X) in the 1990s used cooperative scheduling, which introduced the problem that the whole computer freezes if a single process is implemented badly.

Notes:

[illegible]

- ▶ we locate the **command line interface** (CLI) of our OS
- ▶ we use it to start a **process that executes a simple program**
- ▶ we use operating system tools to **check running processes**
- ▶ we inspect **machine code instructions** contained in an executable file

see directory 03-01 in gitlab repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_WiSe_CS_Practice

Notes:

In summary

- ▶ we have taken a **top-down approach** to study the architecture of modern computers
- ▶ we investigated how “programs” are executed by the CPU at the **machine-code level**
- ▶ we introduced **key functionality of operating systems**
- ▶ key task of OS is to **isolate processes** and **fairly allocate HW resources** like, e.g., memory and CPU time
- ▶ we explained how multiple processes can be run (virtually) simultaneously by a **multi-tasking OS**

64a:	55	push	%ebp
64b:	48	dec	%eax
64c:	89 e5	mov	%esp,%ebp
64d:	48	dec	%eax
64f:	83 ec 10	sub	\$0x10,%esp
652:	48	dec	%eax
653:	8d 05 ab 00 00 00	lea	0xab,%eax
659:	48	dec	%eax
65a:	89 45 f8	mov	%eax,-0x8(%ebp)
65d:	48	dec	%eax
65e:	8b 45 f8	mov	-0x8(%ebp),%eax
661:	48	dec	%eax
662:	89 c6	mov	%eax,%esi
664:	48	dec	%eax
665:	8d 3d a7 00 00 00	lea	0xa7,%edi
66b:	b8 00 00 00 00	mov	\$0x0,%eax
670:	e8 ab fe ff ff	call	520 <printf@plt>
675:	b8 00 00 00 00	mov	\$0x0,%eax
67a:	c9	leave	
67b:	c3	ret	
67c:	0f 1f 40 00	nopl	0x0(%eax)

open questions

- ▶ how can we write programs that **solve actual problems**?
- ▶ how can we translate **source code that is understandable for humans** to instructions that can be executed by the processor?
- ▶ what are **high-level programming languages**?

Self-study questions

1. List and explain the key components of the Von Neumann architecture for stored-program computers.
2. Investigate which other computer architectures apart from the Von Neumann architecture exist and what are their advantages and disadvantages.
3. How is the Von Neumann architecture implemented in a modern CPU?
4. What is the role of the arithmetic-logical unit (ALU) in a modern CPU?
5. Investigate the functionality of a so-called multiplexer and explain how it can be used in an ALU. Write down the truth table of a 2-to-1 multiplexer with two inputs and a single output.
6. What is the role of the control unit (CU) in a modern CPU?
7. What is stored in the Program Counter (PC) of a CPU?
8. What is the role of the Instruction Register (IR) in a modern CPU?
9. Give an example for a processor architecture that define the format and encoding of machine instructions.
10. What are the key functions of an operating system?
11. Explain how an operating system is able to execute multiple processes simultaneously.
12. Why is a context switch in a multi-tasking OS a costly operation?

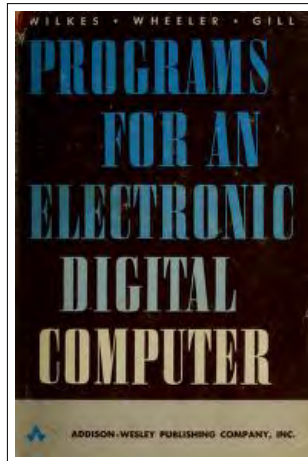
Notes:

Notes:

Literature

reading list

- ▶ M Wilkes, D Wheeler, S Gill: **The Preparation of Programs for an Electronic Digital Computer**, Addison-Wesley, 1951
- ▶ AS Tanenbaum, A Todd: **Structured Computer Organization**, Pearson, 2013
- ▶ D Patterson, JL Hennessy: **Computer Organization and Design: The Hardware/Software Interface**, Morgan Kaufmann, 2013
- ▶ AS Tanenbaum, H Bos: **Modern Operating Systems**, Pearson, 2014



Notes:

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes
 Chair of Machine Learning for Complex Networks
 Center for Artificial Intelligence and Data Science (CAIDAS)
 Julius-Maximilians-Universität Würzburg
 Würzburg, Germany
ingo.scholtes@uni-wuerzburg.de

Julius-Maximilians-
**UNIVERSITÄT
WÜRZBURG**

**Lecture 04
Programming Languages**
 November 12, 2024

Motivation

- we have taken a **top-down approach** to study the hardware/software interface
- we investigated how programs are executed at the **level of machine code**
- we introduced key functionality of **operating systems** and discussed the abstraction of **processes**
- we discussed how multi-tasking allows to execute **multiple processes simultaneously**

open questions

- how can **humans interact with the operating system**?
- how can we write programs that **solve actual problems**?
- how can we translate **code that is understandable for humans** to instructions that can be executed by the CPU?

```

64a: 55          push    %ebp
64b: 48          dec     %eax
64c: 89 e5       mov     %esp,%ebp
64e: 48          dec     %esp
64f: 83 ec 10    sub     $0x10,%esp
652: 48          dec     %eax
653: 8d 05 ab 00 00 00    lea     0xab,%eax
659: 48          dec     %eax
65a: 89 45 f8    mov     %eax,-0x8(%ebp)
65d: 48          dec     %eax
65e: 8b 45 f8    mov     -0x8(%ebp),%eax
661: 48          dec     %eax
662: 89 c6       mov     %eax,%esi
664: 48          dec     %eax
665: 8d 3d a7 00 00 00    lea     0xa7,%edi
66b: b8 00 00 00 00    mov     $0x0,%eax
670: e8 ab fe ff ff    call    520 <printf@plt>
675: b8 00 00 00 00    mov     $0x0,%eax
67a: c9          leave   %eax
67b: c3          ret
67c: 0f 1f 40 00    nopl    0x0(%eax)
  
```

A simple Hello World program in machine code

Notes:

- Lecture L04: Programming Languages** 12.11.2024
- Educational objective:** We introduce high-level programming languages and explain the difference between compiled and interpreted languages.
 - OS User Interfaces
 - Machine Instructions and Assembly Language
 - High-Level Programming Languages and Compilers
 - Interpreted Languages: Python
- Exercise Sheet 3** due 26.11.2024

Notes:

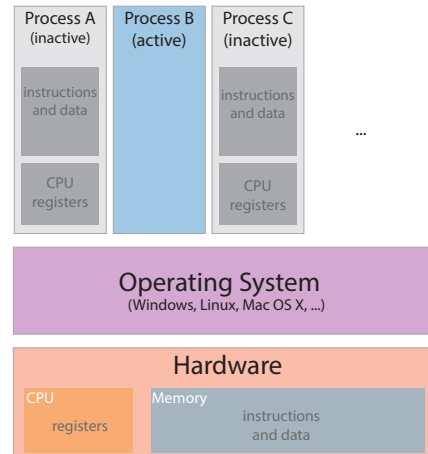
Reminder: Multi-Tasking

- ▶ OS can use **multi-tasking** to execute **multiple processes concurrently** (even on a single CPU)
- ▶ every few milliseconds, OS performs **context switch** between running processes
- ▶ context switch from process *A* to *B* requires to **switch execution context**

context switch from process *A* to *B*

1. interrupt execution of program by CPU
2. save current values in CPU registers (incl. PC) to memory, which fully determine execution state of process *A*
3. restore previously saved CPU registers of process *B* from memory
4. continue execution of program by CPU

- ▶ **OS scheduler** fairly allocates CPU time
- ▶ **preemptive scheduler** forces context switches



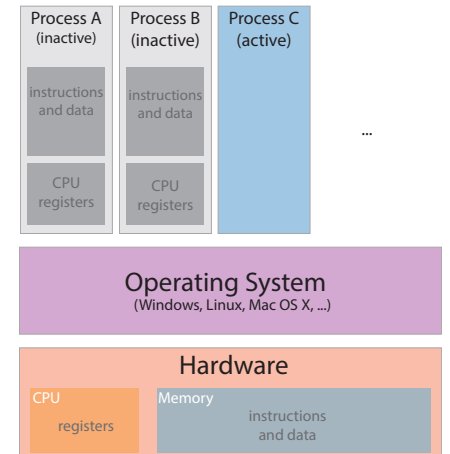
Launching a process

- ▶ we can use OS to **launch a new process** that executes a program
- ▶ reminder: process = **one instance of program** executed by CPU

launching a process

1. OS reads **“executable file”** from hard drive/SSD and copies it into main memory (RAM)
2. “executable file” contains **machine instructions and data**
3. OS sets **program counter of CPU** to address of first machine instruction in main memory
4. OS **transfers control to CPU** (until next context switch)

- ▶ how can we tell OS to launch a new process?



Notes:

1. The opposite of preemptive scheduling is called cooperative scheduling. This means that a context switch can only happen if a process “voluntarily” surrenders the CPU periodically, such that another process can take over. Early operating systems like Windows (before Windows 95) or Mac OS (before Mac OS X) in the 1990s used cooperative scheduling, which introduced the problem that the whole computer freezes if a single process is implemented badly.

Notes:

Graphical User Interfaces (GUI)

- ▶ modern operating systems provide an intuitive and human-friendly **graphical user interface (GUI)**
- ▶ key functions of OS (e.g. launching a process) can be accessed in an intuitive way (e.g. by double-clicking program icon with the mouse)
- ▶ OS provides special program (e.g. file explorer or finder) to **manage files on permanent storage** (hard drive, SSD) or network shares
- ▶ multi-tasking is typically represented by **multiple program windows or icons** that represent **running processes**



definition
A **graphical user interface (GUI)** provides access to the functions of a program or OS by allowing the user to manipulate visual icons and indicators, typically by means of a touch pad, touch screen, or mouse.

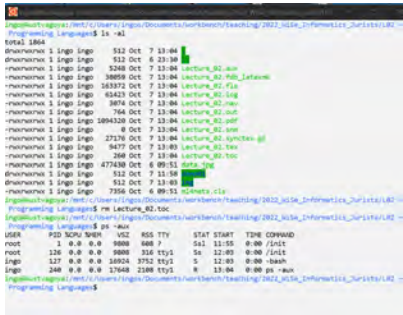
Command line Interfaces

- ▶ in addition to GUI, all major operating systems provide **text-based command line interfaces (CLI)**
 - ▶ **Windows:** command line/PowerShell
 - ▶ **Linux/Mac OS X:** terminal
- ▶ CLI provides **full access to all functions** of an OS

exemplary commands (Linux-based OS)

command	meaning
cd	change directory
ls	list files in current directory
rm	remove file or directory
mv	move/rename file or directory
ps	list running processes
./<executable>	launch new process for program

- ▶ **command-line interpreter** executes commands
- ▶ CLI can be **programmed via “scripts”** (commands in text file)



Command line interface of Ubuntu Linux

definition
A **command-line interface (CLI)** accepts text-based commands to launch and manage processes, manage files, or update system settings.

Notes:

Notes:

Practice Session

- ▶ we locate the **command line interface** (CLI) of our OS
- ▶ we use the CLI to launch a **process that executes a simple HelloWorld program**
- ▶ we use GUI- and CLI-based tools to **monitor and kill running processes**
- ▶ we use the Linux-based CLI-tool objdump to inspect **machine code instructions** contained in an executable file

```
Contents of section .text:
0550 31e40899 d15e4889 324883e4 0959544c 3.E..M...H...PTL
0554 50952a01 0000488d 8d130100 004883d3 .....M.....H..
0558 e6000000 ff15860a 2000f40f 1f440000 .....D.....
0560 488d3da9 0a200055 488d05a1 0a200048 H...UH....H
0570 39f84889 e5741948 0005390a 20004885 9.H..t.H..Z...H.
0580 c074035d ff0e662e 0f1f8400 00000000 ..t..f.....
0590 5dc30f1f 4000662e 0f1f8400 00000000 ]..B.f.....
05a0 488d3d69 0a200048 8d35620a 20005548 H..i..H.5b..UH
05b0 29f84889 e54dc1fe 034809f0 48c1083f )H..H...H..H..?
05c0 4801c648 d1f67418 488d0521 0a200048 H..H..t.H..i..H
05d0 85c0740c 5dff0e66 0f1f8400 00000000 ..t..f.....
05e0 5dc30f1f 4000662e 0f1f8400 00000000 ]..B.f.....
05f0 803d190a 20000075 2f48833d 77092000 .w...u/H.w...
0600 00554889 e5740c48 0d35f609 2000488d .UH..t.H.w...
0610 ffffffab 48ffffff c005f109 2000015d ...H.....D..
0620 c30f1f00 00000000 f3c3660f 1f440000 .....F.D...
0630 554889e5 5de966ff ffff5548 89e54883 UH..f...UH..H.
0640 ec10488d 05950000 00480945 f8480945 .H.....H.E.H.E
0650 f84809c7 e0b7f6ff ffb00000 0000c3c3 .H.....
0660 41574156 4989d741 5541544c 8d254607 AAAVI..AAVTL.XF.
0670 20005548 8d254607 20005541 89f54989 .UH..F..SA..I.
0680 f64c29e5 4883e008 48c1f003 e857f6ff .L)H...H...M..
0690 ff4885ed 74203120 0f1f8400 00000000 .A..t..i.....
06a0 4c09fawc 89f6a489 e741ff14 d04883c3 L..L..D..A...H..
06b0 014839dd 75ea4883 c4083b5d 415c415d .H0.u.H...[[JAAV
06c0 415e415f c390662e 0f1f8400 00000000 A"A..f.....
06d0 f3c3 ..
```

practice session

see directory 04-01 in gitlab repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Programming in machine language?

- ▶ machine code is designed to **make execution by CPU as fast as possible**
- ▶ machine code is **not optimized to be written or read by humans**
- ▶ requires us to manually **address registers**, store values at **addresses in memory**, remember **cryptic machine instructions**, etc.
- ▶ machine code is specific to CPU architecture, i.e. programs in machine code are **not portable**

```
64a: 55          push  %ebp
64b: 48          dec   %eax
64c: 89 e5       mov   %esp,%ebp
64d: 48          dec   %eax
64e: 83 ec 10    sub   $0x10,%esp
64f: 48          dec   %eax
650: 8d 05 ab 00 00 00 lea   0xab,%eax
651: 48          dec   %eax
652: 89 45 f8    mov   %eax,-0x8(%ebp)
653: 48          dec   %eax
654: 8b 45 f8    mov   -0x8(%ebp),%eax
655: 48          dec   %eax
656: 89 c6       mov   %eax,%esi
657: 48          dec   %eax
658: 8d 3d a7 00 00 00 lea   0xa7,%edi
659: b8 00 00 00 00 mov   $0x0,%eax
65a: e8 ab fe ff ff call  520 <printf@plt>
65b: b8 00 00 00 00 mov   $0x0,%eax
65c: c9          leave %eax
65d: c3          ret
65e: 0f 1f 40 00 nopl  0x0(%eax)
```

A simple Hello World program in machine code

challenges

1. how can we make programming **simple and (actually) enjoyable** for human programmers?
2. how can we write **portable programs** that are independent of the processor architecture?

Notes:

Notes:

Assembly language

- ▶ assembly language is a **low-level language** that simplifies writing of machine code
- ▶ different from machine code, assembly language allows **symbolic labels, directives, and comments**
- ▶ **assembler** (software) translates assembly program to machine instructions
- ▶ strong but not strict **correspondence between assembly language and machine instructions**
- ▶ developer maintains **control over machine** instructions, i.e. programs are (potentially) very fast
- ▶ but: assembly code is still **not portable**



Motorola 68000 assembler program

image credit: Wikipedia, public domain

High-Level Languages

- ▶ idea: use **programming language** with **higher-level abstractions** that are easy to understand by humans
- ▶ high-level languages typically provide (at least) the following **abstractions**
 - ▶ **symbolic variables** (with data types), e.g. `int k = 42`
 - ▶ **complex types and data structures** (text, list, queue, etc.)
 - ▶ **control structures** to influence **control flow** in a program
 - ▶ **functions or routines** that can be called for code reuse
- ▶ **compiler** (software) **translates program in high-level language** to simpler machine instructions
 - ▶ original program = **source code**
 - ▶ compiled program = **executable** or **binary**
- ▶ many compilers can generate **binaries for multiple processor architectures** (cross-compilation)

```
int k = 1;
int l = 1;
```

```
for (int i=0; i<10; i++) {
    int t = k + l;
    k = l;
    l = t;
}
```

```
char* text = "Result: %s\n";
printf(text, l);
```

Notes:

Notes:

11

The C programming language

- ▶ **general-purpose programming language** created by Ritchie and Thompson in 1972 as successor to language B
- ▶ one of the most **important and widely-used** programming languages
- ▶ **statically-typed language**, i.e. we must specify type of variable
- ▶ C compilers support **virtually any processor architecture**

limitations of C

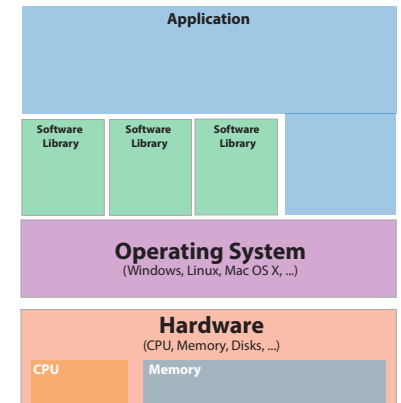
- ▶ error-prone dynamic allocation/release of memory
- ▶ lack of object-oriented abstractions
- ▶ basis for object-oriented “successors” C++ (1979) and Objective-C (Apple, 1984)

```
#include <stdio.h>
#include <unistd.h>

int main(void) {
    char* text = "Hello World!";
    printf("%s\n", text);
    sleep(5);
}
```

Software libraries

- ▶ **self-contained programs** must implement all functions that are needed by the software that we want to develop
- ▶ analogy: if you write a book, you can rely on (and refer to) **common knowledge** published by other authors
- ▶ **software libraries** contain common functionality that can be reused by other programs
 1. binary libraries with machine instructions
 2. library with reusable source code
- ▶ most high-level programming languages provide **standard libraries for common tasks**
 - ▶ complex mathematical operations
 - ▶ reading/writing from/to files
 - ▶ network communication
 - ▶ graphics and visualization



Notes:

Notes:

Application Programming Interfaces

- ▶ software library provides **application programming interface (API)** that enables us to access common functions
- ▶ analogy: table of contents in a book, which gives page number for each “topic”
- ▶ API specifies details that are required to **call function**
 - ▶ **name of function**
 - ▶ number, type and semantics of **parameters** that caller must provide
 - ▶ semantics and type of **return value** that is returned by the function
- ▶ example 1: C library `stdio` provides function `printf` that **outputs text via CLI**
- ▶ example 2: python module `math` provides function `sqrt` that returns **square root of given value**

```
int char *getopt(int, char * const[], const char *); (LEGACY)
int char *gets(char *);
int int getw(FILE *);
int int pclose(FILE *);
void void perror(const char *);
FILE *popen(const char *, const char *);
int int printf(const char *, ...);
int int putchar(int);
int int putchar_unlocked(int, FILE *);
int int putchar_unlocked(int);
int int puts(const char *);
int int puts_unlocked(FILE *);
int int remove(const char *);
int int rename(const char *, const char *);
void void rewind(FILE *);
int int scanf(const char *, ...);
void void setbuf(FILE *, char *);
int int setvbuf(FILE *, char *, int, size_t);
int int sprintf(char *, size_t, const char *, ...);
int int sprintf(char *, const char *, ...);
int int sscanf(const char *, const char *, int ...);
char *strncpy(const char *, const char *, int);
FILE *tmpfile(void);
char *tmpnam(char *);
int int ungetc(int, FILE *);
int int vfprintf(FILE *, const char *, va_list);
int int vprintf(const char *, va_list);
int int vsnprintf(char *, size_t, const char *, va_list);
int int vsprintf(char *, const char *, va_list);
```

excerpt of API of C Standard Library `stdio`

Practice Session

- ▶ we write a simple program in the **high-level language C**
- ▶ we use two **library functions** to print text and to pause the program execution
- ▶ we use the compiler `gcc` to **compile the source code to an executable program**

```
#include <stdio.h>
#include <unistd.h>
```

```
int main(void) {
    char* text = "Hello World!";
    printf("%s\n", text);
    sleep(5);
}
```

practice session

see directory 04-02 in `gitlab` repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

Compiled vs. interpreted languages

- ▶ **compiler translates program** in high-level language to machine code **before** it can be executed
 - ▶ compiled binaries are not portable
 - ▶ users may need to compile source code
 - ▶ each change requires to recompile source code
- ▶ **interpreter can directly execute instructions** in a high-level programming language
- ▶ interpreter is program that reads and executes source code, i.e. process = **instance of interpreter that executes code in a file**
- ▶ no need for (re)compilation, no non-portable binaries
- ▶ interpreted languages are **typically slower than compiled languages** (but not necessarily)

definition

An **interpreter** is a software that directly executes instructions written in a programming language, without requiring its prior compilation to machine code.

Notes:

Introducing Python

- ▶ python is the most **popular interpreted programming language**
- ▶ widely-used for **data processing, analytics, and machine learning**
- ▶ object-oriented with **automatic memory management**, i.e. memory is automatically allocated and released
- ▶ dynamically-typed language, i.e. types of variables are automatically inferred (and can change) at runtime
- ▶ user-friendly, great for **beginners in programming**
- ▶ **rich ecosystem of software libraries** (modules) that implement almost any imaginable functionality



Guido van Rossum, developer of python

image credit: Wikipedia, Doc Searls, CC BY-SA 2.0

Notes:

Basic python syntax

- ▶ python programs are stored in text files (typically with extension .py)
- ▶ **one line in text file = one instruction**

key python statements

- ▶ assignment (=) used to **assign value to a variable**
- ▶ def used to define a **function**
- ▶ import statement used to import functions from modules
- ▶ if and else used to **conditionally execute instructions**
- ▶ for and while used to **repeatedly execute instructions in a loop**

- ▶ “blocks” of instructions grouped by **indentation level**
- ▶ python is **whitespace-sensitive**, i.e. placement of newline, space or tab characters changes semantics
- ▶ python enforces meaningful formatting of code, making programs easy to read for humans

```
import time

def main():
    for i in range(5):
        text = "Hello World!"
        print(text)
        text = 42
        print(text)
    sleep(5)
```

Practice Session

- ▶ we install the Open Source **python distribution Anaconda**
- ▶ we write a simple “Hello World” program in python
- ▶ we use the python interpreter to execute our program
- ▶ we **inspect running processes** during the execution of our program

```
import time

def main():
    text = "Hello World!"
    print(text)
    text = 42
    print(text)
    sleep(5)
```

practice session

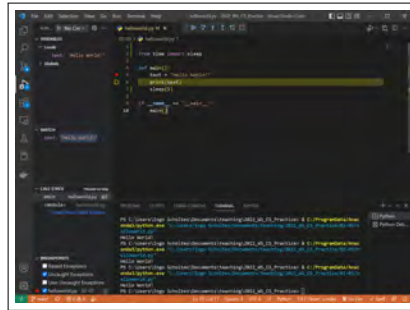
see directory 04-03 in **gitlab** repository at
→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

Integrated Development Environment (IDE)

- ▶ all we need to write python program is **text editor** and **python interpreter** (i.e. executable `python.exe`)
- ▶ sufficient for small single-file programs
- ▶ what about **complex software with hundreds of files and millions of lines in code?**
- ▶ integrated development environments (IDEs) are specialized tools to **support and simplify development of complex software**
- ▶ IDEs provide advanced functions to edit and format code, semantically highlight/color keywords, compile and/or execute program, and find errors



Open Source IDE Visual Studio Code

definition

An **Integrated Development Environment (IDE)** is a software that simplifies the programming of computers. It minimally provides functions to **edit source code files**, compile and/or execute programs, and **find errors at compile- and run-time**.

Practice Session

- ▶ we use the **integrated development environment (IDE)** Visual Studio Code to write and execute a simple python program
- ▶ we use VS Code to **rename variables and refactor code**
- ▶ we use the **debugger of Visual Studio Code** for a step-wise execution of python statements

```
import time

def main():
    text = "Hello World!"
    print(text)
    text = 42
    print(text)
    sleep(5)
```

practice session

see directory 04-04 in **gitlab** repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

In summary

- ▶ we inspected the GUI and the CLI of modern operating systems
- ▶ we motivated the use of high-level programming languages
- ▶ we explained the difference between compiled and interpreted languages
- ▶ we introduced the popular interpreted high-level language python and wrote a first program

```
64a: 55      push  %ebp
64b: 48      dec   %eax
64c: 89 e5   mov   %esp,%ebp
64e: 48      dec   %eax
64f: 83 ec 10 sub   $0x10,%esp
652: 48      dec   %eax
653: 8d 05 ab 00 00 00 lea    0xab,%eax
659: 48      dec   %eax
65a: 89 45 f8 mov   %eax,-0x8(%ebp)
65d: 48      dec   %eax
65e: 8b 45 f8 mov   -0x8(%ebp),%eax
661: 48      dec   %eax
662: 89 c6   mov   %eax,%esi
664: 48      dec   %eax
665: 8d 3d a7 00 00 00 lea    0xa7,%edi
66b: b8 00 00 00 00 mov   $0x0,%eax
670: e8 ab fe ff ff call   520 <printf@plt>
675: b8 00 00 00 00 mov   $0x0,%eax
67a: c9      leave
67b: c3      ret
67c: 0f 1f 40 00 nopl   0x0(%eax)
```

open issues

- ▶ how can we use high-level languages to solve actual problems?
- ▶ what are algorithms and how we can we implement them?
- ▶ need to develop algorithmic thinking, which is key to understand how computer scientists think and work.

Self-study questions

1. Explain the difference of a GUI and a CLI of an operating system. Which one is more intuitive? Which one is more powerful?
2. Explain the steps taken by an OS to launch a process that executes a HelloWorld program stored in an executable file.
3. Explain the difference between machine instructions and assembler code.
4. What are the advantages of high-level programming languages like C compared to assembler?
5. List abstractions provided by a high-level programming language that are not provided by machine instructions?
6. What is a variable in a high-level language?
7. What is the difference between statically- and dynamically-typed programming languages?
8. What is a compiler and what is an interpreter?
9. Explain the steps needed to write and execute a Hello World program written in the programming language C.
10. Explain the steps needed to write and execute a Hello World program written in the programming language python.
11. What are advantages/disadvantages of compiled and interpreted programming languages?
12. What advantages does an integrated development environment (IDE) provide?

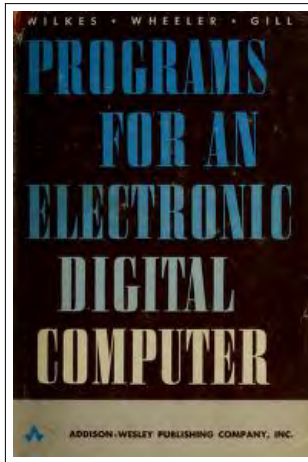
Notes:

Notes:

Literature

reading list

- ▶ W Kernighan, D Ritchie: **The C Programming Language**, Prentice Halle, 2000
- ▶ F Kaefer, P Kaefer: **Introduction to Python Programming for Business and Social Science Applications**, SAGE Publications, 2020



Notes:

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

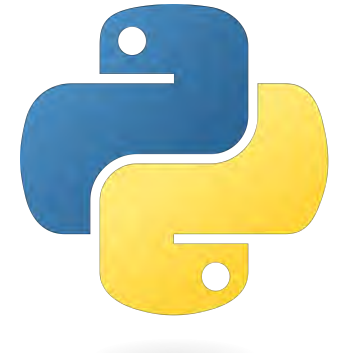
ingo.scholtes@uni-wuerzburg.de

Motivation

- ▶ we introduced **high-level programming languages** that are translated to machine code
- ▶ we wrote a first “Hello World” program in C and inspected the machine code generated by the **compiler**
- ▶ we distinguished between **compiled languages** like C/C++ and **interpreted languages** like python
- ▶ we introduced basics of the **python syntax**

open issues

- ▶ how to write programs that **solve actual problems**?
- ▶ what are **algorithms** and how we can we implement them in high-level languages like python?
- ▶ need to develop **algorithmic thinking**, which is key to understand how a computer (scientist) works



Notes:

- **Lecture L05: Algorithmic Thinking** 19.11.2024
- **Educational objective:** We introduce algorithms for basic problems like binary search and sorting. We discuss the runtime of algorithms and cover basic data structures.
 - Python Data Structures
 - A Simple Algorithm
 - Binary Search Algorithm
 - Basic Sorting Algorithms
- **Exercise Sheet 4** due 26.11.2024

Notes:

Recap: python Syntax

- ▶ python programs are stored in text files (typically with extension .py)
- ▶ **one line in text file = one instruction**

key python statements

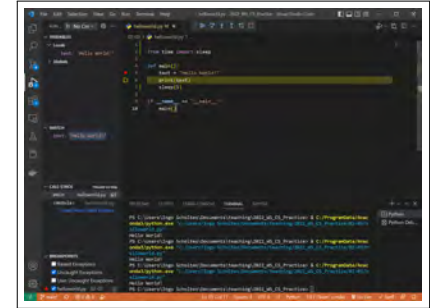
- ▶ assignment (=) used to **assign value to a variable**
- ▶ def used to define a **function**
- ▶ import statement used to import functions from modules
- ▶ if and else used to **conditionally execute instructions**
- ▶ for and while used to **repeatedly execute instructions in a loop**
- ▶ “blocks” of instructions grouped by **indentation level**
- ▶ python is **whitespace-sensitive**, i.e. placement of newline, space or tab characters changes semantics
- ▶ python enforces meaningful formatting of code, making programs easy to read for humans

```
import time

def main():
    for i in range(5):
        text = "Hello World!"
        print(text)
        text = 42
        print(text)
    sleep(5)
```

Integrated Development Environment (IDE)

- ▶ all we need to write python program is **text editor** and **python interpreter** (i.e. executable python.exe)
- ▶ sufficient for small single-file programs
- ▶ what about **complex software with hundreds of files and millions of lines in code?**
- ▶ integrated development environments (IDEs) are specialized tools to **support and simplify development of complex software**
- ▶ IDEs provide advanced functions to edit and format code, semantically highlight/color keywords, compile and/or execute program, and find errors



Open Source IDE Visual Studio Code

definition

An **Integrated Development Environment (IDE)** is a software that simplifies the programming of computers. It minimally provides functions to **edit source code files**, compile and/or execute programs, and **find errors at compile- and run-time**.

Notes:

- Note that the “grammatical structure” of a (programming) language is called “syntax”, which contains the Greek words “syn” (together) and “taxis” (ordering/composition). The syntax of a language defined keywords and determines the ordering of characters that constitutes a valid sentence or (program) in a (programming) language.

Notes:

Python Data Structures

- ▶ all programming languages support **basic data types**
 - ▶ integer numbers, i.e. 42, -55, 0
 - ▶ floating point numbers, i.e. 4.52, 1.567e2, 2.0e - 2
 - ▶ character types, i.e. "c", "t"
 - ▶ string types, i.e. "Lecture"
- ▶ python is a **dynamically-typed language**, i.e. we can assign any type to a variable
- ▶ what if we need more **complex structures to store data**?
 - ▶ list of numbers
 - ▶ all sentences of a book
 - ▶ mapping from numbers to text
 - ▶ queue of jobs to be executed in sequence
- ▶ python standard library provides **complex data types** that can hold list, sequences, dictionaries of values

Python Lists

- ▶ **python lists** can hold **ordered sequence of elements** of any type

adding / removing elements

- ▶ append allows to **append additional values** at end of list
- ▶ pop can be used to **remove and return element** at a given index (or at the end)
- ▶ remove **deletes first occurrence of a value**

- ▶ we can use zero-based **integer indexing** to read/write elements at specific position
- ▶ **slicing operator** [start:end:step] can be used to return new list with selected elements
- ▶ using append and pop(0) **we can use list as queue**, where elements are returned in **first-in-first-out (FIFO) order**

```
# create list
l = [2, 42, 120, 18, 420]

# adding/removing elements
l.append('hello')
l.remove(120)
print(l.pop())
print(l.pop(0))

# index-based access
l[1] = 43

# elements up to index 2
# (excluding 2)
print(l[:2])

# elements starting from index 1
# (including 1)
print(l[1:])

# initialize list with 42 zero entries
l = [0]*42
```

Notes:

Notes:

Python Tuples

- ▶ by appending, assigning or removing elements, **python lists can dynamically change their size** and **elements can change** during lifetime of list
- ▶ requires complex implementation that makes some operations relatively slow
- ▶ for **fixed-size ordered sequences that cannot change**, we can use python tuples
- ▶ **indexing and slicing** works the same as for lists
- ▶ elements cannot change and size of tuple cannot grow or shrink
- ▶ we can use + operator to **concatenate two tuples**, returning a new tuple

```
# create tuple
t = (2, 42, 120, 189, 420)

# index-based access
print(t[1])

# slicing
print(t[:2])
print(t[1:])

# NOT VALID
t[1] = 43

# returns new tuple with additional
# elements
t2 = t + (4,5,6)
print(t2)
```

Python Sets

- ▶ lists and tuples are **ordered sequences**
- ▶ checking whether an **element is in a list/tuple** requires to **test all elements** → naive linear search
- ▶ for **unordered collection of objects without duplicates** we can use python set
- ▶ useful to **eliminate duplicate elements** and **quickly test for membership**
- ▶ **python sets** are unordered and thus **do not support indexing or slicing**

```
# create set
s = {2, 'hello', 120, 42, 189, 420}
print(s)

# check membership
print('hello' in s)

# add element
s.add(32)

# remove element
s.remove('hello')

# NOT VALID
s[1]
s[:4]
```

Notes:

Notes:

Python Dictionaries

- ▶ we often need an **associative mapping** that maps **unique keys** to values
 - ▶ string/number as unique identifier (key)
 - ▶ arbitrary data of record (value)
- ▶ useful to **quickly find data** that are stored under a given key
- ▶ we can read/write entries using **index syntax** similar to lists
- ▶ but: index does not need to be an integer
- ▶ **reverse lookup** (i.e. find key(s) for a given value) not supported

```
# create dictionary
d = { 'hello': 'Hallo',
      'world': 'Welt',
      'teacher': ['Ingo', 'Scholtes']
}

# check membership of key
print('hello' in d)

# access value of given key
print(d['hello'], d['teacher'])

# assign value to (new) key
d['audience'] = 'Studierende'
```

Practice Session

- ▶ we show how to install the Open Source **python distribution Anaconda**
- ▶ we use the **integrated development environment** (IDE) Visual Studio Code to write and execute a simple python program
- ▶ we use VS Code to **rename variables and refactor code**
- ▶ we use the **debugger of Visual Studio Code** for a step-wise execution of python statements
- ▶ we demonstrate lists, tuples, sets, and dictionaries in python

```
import time

def main():
    text = "Hello World!"
    print(text)
    text = 42
    print(text)
    sleep(5)
```

practice session

see directory 05-01 in gitlab repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

What is an algorithm?

definition → L01 - Motivation

An **algorithm** is a sequence of precisely defined (mathematical) instructions that must be executed to solve a given problem.

- ▶ algorithm takes a (possibly empty) **input** and produces – after a **finite number of steps** – a desired **output**
- ▶ expressing an algorithm in terms of a **programming language** allows us to **implement** it on a computer

Example: pencil-and-paper algorithm to add two numbers

- step 1 start at right-most position
- step 2 add digits at current position
- step 3 write last digit of sum below current position
- step 4 for sums ≥ 10 additionally **carry over 1** to position on the left
- step 5 move one position to left and go to step 2

$$\begin{array}{r} 1 \quad 2 \quad 5 \quad 7 \\ 2 \quad 1 \quad 9 \quad 3 \quad 2 \\ \hline 4 \quad 1 \quad 8 \quad 9 \end{array}$$

Group Exercise 05-01

- ▶ Assume that we want to implement the **pen-and-pencil algorithm to add two decimal numbers with an arbitrary number of digits**. Specify a reasonable input and output of this algorithm.

- ▶ input: **two python lists**, containing the digits of two decimal numbers
- ▶ output: **a python list** containing the digits of the sum

- ▶ Develop a python function add that implements the pen-and-pencil algorithm, using control structures like while, for, if as well as a python list

```
def add(a, b):
    l = len(a)
    sum = [0]*l
    carry = 0
    i = l-1
    while i >= 0:
        s = a[i] + b[i] + carry
        if s >= 10:
            carry = 1
            s = s - 10
        else:
            carry = 0
        sum[i] = s
        i = i-1
    return sum
```

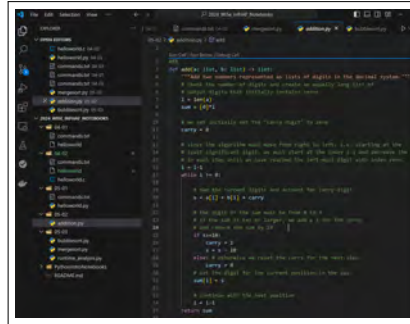
Notes:

Notes:

- In lecture L02 we have seen how we can use digital logics to implement the addition of two binary numbers in terms of hardware. Thanks to the fact that this operation is implemented in the ALU of the CPU, we can directly add two 32 or 64 bit numbers by a single machine instruction (e.g. ADD). This implies that we can use the ADD operator + in high-level languages like python, which is directly mapped to this machine instruction.
- As an exercise, we pretend that there was no such operation that allows to add numbers with more than one digit. Let us implement the **pen-and-pencil algorithm to add decimal numbers** in python.
- Note that such an algorithm can still be useful if we want to add two numbers that cannot be represented by 64 bits or less, which may not be supported by the ALU of a common CPUs.

Practice Session

- ▶ we use lists to **implement the algorithm** developed in the previous group exercise in python
- ▶ we **test our algorithm** with different inputs

A screenshot of a code editor with a dark theme. The left sidebar shows a file explorer with a directory structure. The main editor area contains Python code. The code defines a function 'find' that takes a list 'l' and a target 't' as arguments. It uses a loop to iterate through the list and returns the index of the first element that matches the target. The code is as follows:

```
def find(l, t):  
    for i in range(len(l)):  
        if l[i] == t:  
            return i  
    return -1  
  
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
t = 5  
print(find(l, t))
```

practice session

see directory 05-02 in gitlab repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Search problems

- ▶ we are frequently confronted with **standard problems** that can be solved by well-understood **standard algorithms**
- ▶ exemplary standard problem: search for an object

search problems

Search problems are a class of problems that seek to quickly **find a given object** within a certain **data structure**.

- ▶ examples
 - ▶ problem 1: search name “Turing” in arbitrary list of 10,000 names
 - ▶ problem 2: search name “Turing” in a phonebook
- ▶ optimal solution to the search problem depends on **prior knowledge on the data structure**



image credit: DALL-E generated image, prompt “needle in a haystack”

Notes:

Searching in sorted data

- ▶ we can **sort a list of objects** whenever for pair of objects a and b we can determine whether $a \geq b$
- ▶ we assume that we search object x in a **list sorted in ascending order**, i.e. list l where for index i we have

$l[i + 1] \geq l[i]$

- ▶ **naive algorithm** checks $x == l[i]$ for index $i = 0, 1, \dots$

binary search algorithm

- ▶ test if x is larger/smaller/equal than middle element c
- ▶ if $x == c$ return object
- ▶ if $x > c$ repeat search in elements **right of c**
- ▶ if $x < c$ repeat search in elements **left of c**

- ▶ binary search is example for **divide-and-conquer algorithm**

- ▶ both algorithms give correct result, but **which one is “better”?**

searching for $x = 17$

sorted list

2	3	5	11	17	23	47
---	---	---	----	----	----	----

step one

2	3	5	11	17	23	47
-	-	-	-	17	23	47

step two

-	-	-	-	17	23	47
-	-	-	-	17	-	-

step three

-	-	-	-	17	-	-
---	---	---	---	----	---	---

found $x = 17$!

Complexity of binary search

1/2

we can evaluate algorithms in terms of **computational complexity**, i.e. we count how many steps they **maximally require** to produce the correct output for a given input?

example 1: how many steps do we need in list l with **32 objects**

naive (linear) search algorithm

step	tested element
1	0
2	1
3	2
...	...
32	31

requires **32 steps for 32 objects**

binary search algorithm

step	tested element
1	16
2	8 or 24
3	4, 12, 20, or 28
4	2, 6, 10, 14, 18, 22, 26, or 30
5	0, 1, 3, 5, 7, 8, 9, 11, 13, 15, 16, 17, ... or 31

requires **5 steps for 32 objects**

Notes:

Notes:

Complexity of binary search

2/2

we can evaluate algorithms in terms of **computational complexity**, i.e. we count how many steps they **maximally require** to produce the correct output for a given input?

example 2: how many steps do we need in list / with **64 objects**

naive (linear) search algorithm

step	tested element
1	0
2	1
3	2
4	3
...	...
63	64

requires **64 steps for 64 objects**

binary search algorithm

step	tested element
1	32
2	16 or 48
3	8, 24, 40, or 56
4	4, 12, 20, 28, ... or 60
5	2, 6, 10, 14, 18, 22, ... or 62
6	0, 1, 3, 5, 7, 9, 11, 13, 15 ... or 63

requires **6 steps for 64 objects**

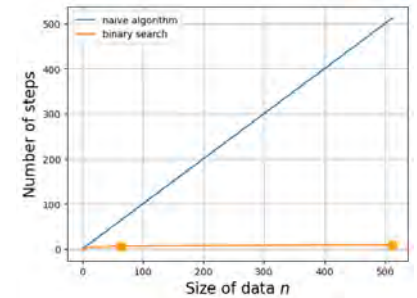
Linear vs. logarithmic complexity

- ▶ naive search algorithm requires one step for each entry in the input list, i.e. runtime is **proportional to the input size**
- ▶ we say an algorithm has **linear complexity** if for input with size n it requires at most

$$c + x \cdot n$$

steps for some numbers c and x

- ▶ thanks to sorted input, **binary search algorithm** requires **less than linear** number of steps
- ▶ how does number of steps grow as we increase input size n ?

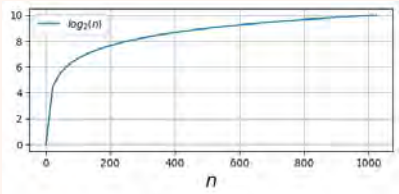


Notes:

Notes:

Logarithms

Logarithm



For a number n , the logarithm $\log_b(n)$ with base b of n is the number x such that $b^x = n$.

logarithms base $b = 2$

n	binary number	$\log_2(n)$
$2^0 = 1$	1	0
$2^1 = 2$	10	1
$2^2 = 4$	100	2
$2^3 = 8$	1000	3
$2^4 = 16$	10000	4

logarithms base $b = 10$

n	$\log_{10}(n)$
$10^0 = 1$	0
$10^1 = 10$	1
$10^2 = 100$	2
$10^3 = 1000$	3
$10^4 = 10000$	4

Complexity of search algorithms

we can evaluate algorithms in terms of **computational complexity**, i.e. we count how many steps they **maximally require** to produce the correct output for a given **input of size n** ?

how many steps do we need in list l with $n = 64$ objects

naive (linear) search algorithm

step	tested element
1	0
2	1
3	2
4	3
...	...
64	63

requires n steps for n objects

binary search algorithm

step	tested element
1	32
2	16 or 48
3	8, 24, 40, or 56
4	4, 12, 20, 28, ... or 60
5	2, 6, 10, 14, 18, 22, ... or 62
6	0, 1, 3, 5, 7, 9, 11, 13, 15 ... or 63

requires $\log_2 n$ steps for n objects

Notes:

- It is easy to see that $\log_b(n)$ increases proportionally with the number of digits of n in a b -nary numeral system

Notes:

Sorting problem

- ▶ for binary search, we assumed that the list of objects is **sorted**
- ▶ to sort objects we must be able to **compare them**, i.e. for each pair a, b we must be able to determine $a \geq b$
- ▶ how can we compare pairs of
 - ▶ numbers,
 - ▶ words,
 - ▶ books,
 - ▶ emojis?

input:

7	2	47	23	5	11
---	---	----	----	---	----

desired output:

2	5	7	11	23	47
---	---	---	----	----	----

sorting problem

The **sorting problem** refers to the problem of sorting a list of **pairwise comparable objects** in ascending or descending order.

- ▶ in the following, we consider the sorting problem for a list of **integer numbers**

BubbleSort algorithm

- ▶ simple idea: repeatedly **compare pairs** of numbers and **swap them** if they are in the wrong order
- ▶ with each swap ...
 - ▶ larger numbers progressively move to right
 - ▶ smaller numbers progressively move to left
- ▶ in each pass of the algorithm, we must compare **all subsequent pairs** of numbers in the list
- ▶ if we have zero swaps during a pass, we know that the list is sorted!
- ▶ in the example, we needed
 - ▶ $4 \cdot 5 = 20$ comparisons
 - ▶ $4 + 2 + 1 = 7$ swaps
- ▶ how many comparisons do we need in **best/worst case**?

third pass

2	7	5	11	23	47
2	7	5	11	23	47
2	7	5	11	23	47
2	5	7	11	23	47
2	5	7	11	23	47
2	5	7	11	23	47
2	5	7	11	23	47
2	5	7	11	23	47
2	5	7	11	23	47
2	5	7	11	23	47

5 comparisons, 1 swap

Notes:

Notes:

Worst-case complexity of BubbleSort

worst-case runtime

For an input **list sorted in reverse order** BubbleSort algorithm requires n passes with $n - 1$ comparisons each.

47	23	11	7	5	2
----	----	----	---	---	---

$$n = 6$$

$$n \cdot (n - 1) = 6 \cdot 5 = 30 \text{ comparisons}$$

best-case runtime

For an input **list that is already sorted** BubbleSort algorithm requires a single pass with $n - 1$ comparisons.

2	7	5	11	23	47
---	---	---	----	----	----

$$n = 6$$

$$n - 1 = 5 \text{ comparisons}$$

Linear vs. polynomial complexity

- ▶ for input list with n elements, BubbleSort has **worst-case runtime** of

$$n \cdot (n - 1) = n^2 - n$$

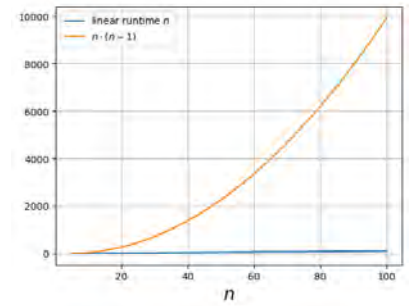
i.e. number of required steps grows as **second power (i.e. square) of input size n**

- ▶ we call expressions of the form

$$a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \dots a_0 \cdot n^0$$

polynomial

- ▶ for polynomials with power larger than one, runtime **grows over-proportionally** with input size



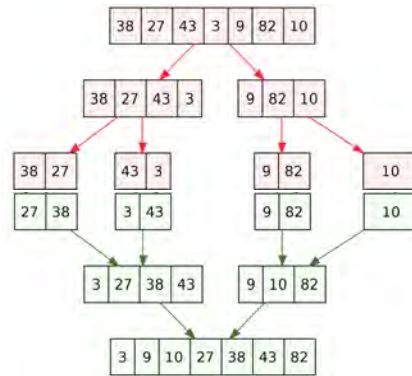
linear vs. polynomial growth of complexity

Notes:

Notes:

MergeSort

- ▶ can we **sort a list faster** than BubbleSort?
- ▶ assume that we have **two already sorted lists** l_1 and l_2
- ▶ in $n = n_1 + n_2$ steps we can **merge** l_1 and l_2 into a new sorted list l
- ▶ we can apply **divide-and-conquer idea** behind binary search to sorting
- ▶ phase 1: repeatedly **split** input until we are left with lists with one element (which are already sorted)
- ▶ phase 2: repeatedly **merge** increasingly large (sorted) lists until full list is sorted

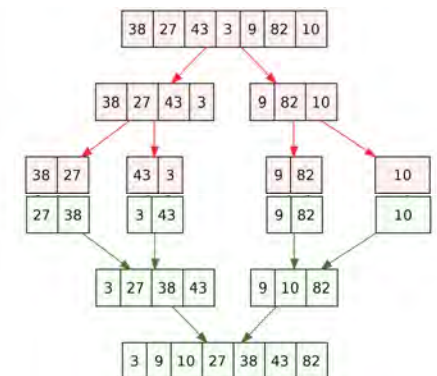


Complexity of MergeSort

- ▶ assume that for a **list with n elements** MergeSort takes $T(n)$ steps
- ▶ each split/merge then requires

$$2 \cdot T\left(\frac{n}{2}\right) + n$$
- ▶ starting with $T(1) = 0$ we have
 - ▶ $T(2) = 2 \cdot T(1) + 2 = 2$
 - ▶ $T(4) = 2 \cdot T(2) + 4 = 2 \cdot 2 + 4 = 8$
 - ▶ $T(8) = 2 \cdot T(4) + 8 = 2 \cdot 8 + 8 = 24$
 - ▶ $T(16) = 2 \cdot T(8) + 16 = 2 \cdot 24 + 16 = 64$
 - ▶ ...
- ▶ we can calculate runtime of MergeSort as

$$T(n) \approx n \cdot \log_2(n)$$

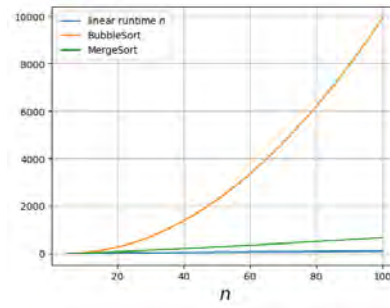


Notes:

Notes:

Complexity of sorting?

- ▶ with BubbleSort we can sort n numbers in $n - 1$ steps in best case and $n \cdot (n - 1)$ in worst case
- ▶ MergeSort improves worst-case complexity of BubbleSort from n^2 to $n \log_2(n)$
- ▶ **on average** MergeSort requires $n \log_2(n)$ steps
- ▶ to sort n objects based on **pairwise comparisons**, there is **no algorithm exist that requires less than $n \log_2(n)$ steps on average**
- ▶ but: there are specialized algorithms to **sort n integer numbers in a fixed range** with linear runtime → self-study questions



worst-case complexity of MergeSort vs. BubbleSort

Practice Session

- ▶ we implement BubbleSort in python
- ▶ we implement the divide-and-conquer method MergeSort
- ▶ we study the **runtime of both algorithms** in increasingly large input lists

```
def mergeSort(l):  
    if len(l) <= 1:  
        return l  
    # split list in two  
    mid = floor(len(l)/2)  
    left = l[:mid]  
    right = l[mid:]  
    # sort left and right  
    left = mergeSort(left)  
    right = mergeSort(right)  
    # merge the two sorted lists  
    return merge(left, right)  
  
def merge(l1, l2):  
    result = []  
    while l1 && l2:  
        if l1[0] < l2[0]:  
            result.append(l1.pop(0))  
        else:  
            result.append(l2.pop(0))  
    while l1:  
        result.append(l1.pop(0))  
    while l2:  
        result.append(l2.pop(0))  
    return result
```

practice session

see directory 05-03 in **gitlab** repository at

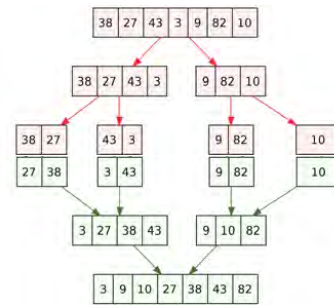
→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

In summary

- ▶ we covered basic **python data structures** like sets, tuples, lists, and dictionaries
- ▶ we introduced **basic algorithms for standard computational problems** like searching and sorting
- ▶ we evaluated the **computational complexity** of sort and search algorithms
- ▶ we highlighted the difference between **logarithmic, linear, and polynomial runtime**



Self-study questions

1. Explain the differences between a set, a tuple, a dictionary and a list in python.
2. Give a formulation of the Pen-And-Pencil algorithm to add two numbers in python and explain it in your own words.
3. Extend the Pen-And-Pencil algorithm from the group exercise such that it can add numbers given as sequences of digits in an arbitrary k -nary numeral system.
4. Give a formulation of the Binary Search algorithm in python and explain it in your own words.
5. Could we generalize the Binary Search algorithm such that in each step we split the list into three equally large parts, which would lead to a runtime $\log_3(n)$?
6. Give a formulation of the BubbleSort algorithm in python and explain it in your own words.
7. Give an example for an input for which the BubbleSort algorithm performs the maximum/minimum number of comparisons.
8. Count the number of swaps in an input list with n elements, where BubbleSort performs the maximum number of comparisons.
9. Give a formulation of the MergeSort algorithm in python and explain it in your own words.
10. Investigate the BucketSort algorithm for integers in a fixed range and explain why it takes less than $n \log_2 n$ steps on average.

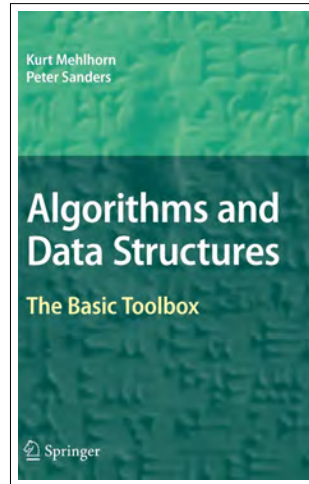
Notes:

Notes:

Literature

References

- ▶ K Mehlhorn, P Sanders: **Algorithms and Data Structures - The Basic Toolbox**, Springer, 2008
- ▶ TH Cormen, CE Leiserson, RL Rivest, C Stein: **Introduction to Algorithms**, MIT Press, 2001
- ▶ F Kaefer, P Kaefer: **Introduction to Python Programming for Business and Social Science Applications**, SAGE Publications, 2020
- ▶ DE Knuth: **The Art of Computer Programming. Vol. 3: Sorting and Searching**, Addison-Wesley, 1998
- ▶ EH Friend: **Sorting on Electronic Computer Systems**, Journal of the ACM, Vol. 3, 1956



Notes:

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

Julius-Maximilians-
**UNIVERSITÄT
WÜRZBURG**

**Lecture 06
Algorithmic Thinking II**

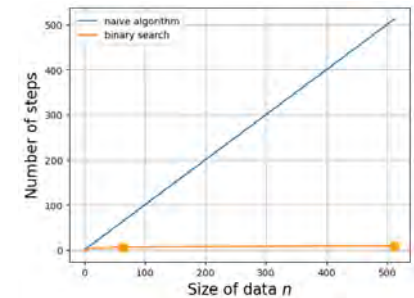
November 26, 2024

Motivation

- ▶ we considered two simple examples that illustrate how we can **use algorithms to solve problems**
- ▶ we reconsider the pencil-and-paper addition algorithm and implemented it in python
- ▶ we motivated the search problem and showed how to address it with the **binary search algorithm**
- ▶ we showed how we can compare the efficiency of algorithms in terms of their **computational complexity**
- ▶ today we continue our introduction to **algorithmic thinking** by addressing the following questions

today's agenda

- ▶ how can we **efficiently sort data**?
- ▶ how can we **encrypt data** and ensure **secure communication**?
- ▶ can we find an **efficient algorithm for any problem**?



computational complexity of **naive search algorithm** vs. **binary search algorithm** in list with size n

Notes:

- **Lecture L06: Algorithmic Thinking II** 26.11.2024
- **Educational objective:** We introduce two basic algorithms for the sorting problem and investigate the runtime of these algorithms. We further discuss basic encryption algorithms and introduce the concept of public-key cryptography.
 - The Sorting Problem
 - Recursion and MergeSort
 - Encryption Algorithms
 - Public-Key Cryptography
- **Exercise Sheet 4** due 03.12.2024

Notes:

Sorting problem

- ▶ **binary search algorithm** assumes list of objects is **sorted** in ascending (or descending) order
- ▶ to sort objects we must be able to **compare them**, i.e. for a pair a, b we must be able to determine $a \geq b$
- ▶ how can we compare pairs of
 - ▶ numbers,
 - ▶ words,
 - ▶ books,
 - ▶ emojis?

sorting problem

The **sorting problem** refers to the problem of sorting a list of **pairwise comparable objects** in ascending or descending order.

- ▶ in the following, we consider the sorting problem for a list of **integer numbers**

input:

7	2	47	23	5	11
---	---	----	----	---	----

desired output:

2	5	7	11	23	47
---	---	---	----	----	----

BubbleSort algorithm

- ▶ simple idea: repeatedly **compare pairs** of numbers and **swap them** if they are in the wrong order
- ▶ with each swap ...
 - ▶ larger numbers progressively move to right
 - ▶ smaller numbers progressively move to left
- ▶ in each pass of the algorithm, we must compare **all subsequent pairs** of numbers in the list
- ▶ if we have zero swaps during a pass, we know that the list is sorted!
- ▶ in the example, we needed
 - ▶ $4 \cdot 5 = 20$ comparisons
 - ▶ $4 + 2 + 1 = 7$ swaps
- ▶ how many comparisons do we need in **best/worst case**?

third pass

2	7	5	11	23	47
2	7	5	11	23	47

2	7	5	11	23	47
2	5	7	11	23	47

2	5	7	11	23	47
2	5	7	11	23	47

2	5	7	11	23	47
2	5	7	11	23	47

2	5	7	11	23	47
2	5	7	11	23	47

5 comparisons, 1 swap

Notes:

Notes:

Worst-case complexity of BubbleSort

worst-case example

For an input **list sorted in reverse order** BubbleSort algorithm requires n passes with $n - 1$ comparisons each.

47	23	11	7	5	2
----	----	----	---	---	---

$$n = 6$$

$$n \cdot (n - 1) = 6 \cdot 5 = 30 \text{ comparisons}$$

best-case example

For an input **list that is already sorted** BubbleSort algorithm requires a single pass with $n - 1$ comparisons.

2	7	5	11	23	47
---	---	---	----	----	----

$$n = 6$$

$$n - 1 = 5 \text{ comparisons}$$

We call the maximum runtime on any input the **worst-case time complexity** or **worst-case computational complexity** of an algorithm.

Linear vs. polynomial complexity

- ▶ for input list with n elements, BubbleSort has **worst-case runtime** of

$$n \cdot (n - 1) = n^2 - n$$

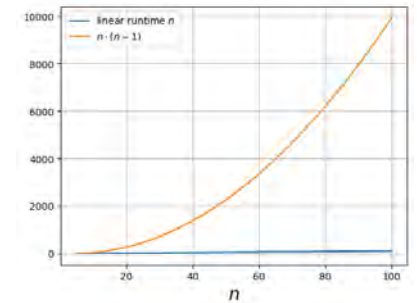
i.e. number of required steps grows as **second power (i.e. square) of input size n**

- ▶ we call expressions of the form

$$a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \dots a_0 \cdot n^0$$

polynomial

- ▶ for polynomials with power larger than one, runtime **grows over-proportionally** with input size



linear vs. polynomial growth of complexity

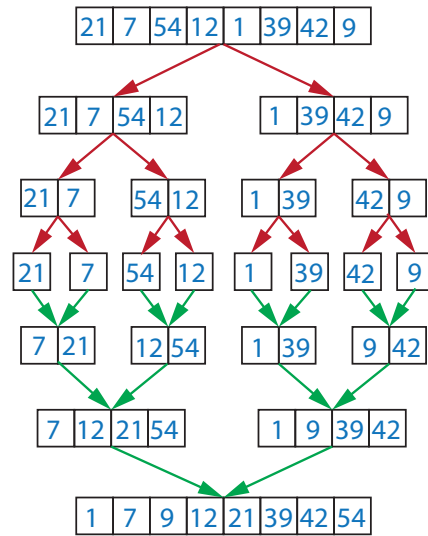
Notes:

- for runtime we typically use the term “time complexity” because there are other aspects of “computational complexity” such as the amount of memory (or disk space) required by an algorithm. This is often called “space complexity” and the term computational complexity then encompasses both time and space complexity.

Notes:

A divide-and-conquer algorithm

- ▶ can we **sort a list faster** than BubbleSort?
- ▶ assume that we have **two already sorted lists** l_1 and l_2 with n_1 and n_2 elements respectively
- ▶ in $n = n_1 + n_2$ steps we can **merge** l_1 and l_2 into a new sorted list l
- ▶ we can apply **divide-and-conquer idea** behind binary search to sorting
- ▶ **phase 1**: repeatedly **split** input until we are left with lists with one element (which are already sorted)
- ▶ **phase 2**: repeatedly **merge** increasingly large (sorted) lists until full list is sorted



MergeSort algorithm

- ▶ how can we implement MergeSort in python
- ▶ assume we have a **function merge** that merges two sorted lists to a new sorted list
- ▶ **function merge_sort** that sorts a list l with n entries must perform the following steps ...
 1. split list in two equally large lists l_1 and l_2
 2. **call itself** on l_1 and l_2
 3. merge sorted lists to result
- ▶ concept of a function calling itself is called **recursion**
- ▶ recursion terminates when list only contains single (or zero) element (which we call **"base case"**)
- ▶ common way to apply **divide-and-conquer**, i.e. function calls itself on smaller problem instances

```
def merge(l1: list, l2: list) -> list
    l = []
    ...
    return l

def merge_sort(l: list) -> list
    if len(l) < 2:
        return l
    mid = math.floor(len(l)/2)
    l1 = l[:mid]
    l2 = l[mid:]
    l1_s = merge_sort(l1)
    l2_s = merge_sort(l2)
    l_s = merge(l1_s, l2_s)
    return l_s
```

python implementation of recursive
MergeSort algorithm

Notes:

Notes:

- the term recursion comes from the Latin word "recurso", which translates to "to come back"

Complexity of MergeSort?

- assume that `merge_sort` requires $T(n)$ steps for **list with n elements**
- in each step, we run `merge_sort` on **two lists with half the size** and then **merge results**
- each step thus requires

$$\underbrace{2 \cdot T\left(\frac{n}{2}\right)}_{\text{recursive calls of merge_sort}} + \underbrace{n}_{\text{merge lists}}$$

- how often do we need to split until we are left with lists with single entry?
- like for binary search, we **need $\log_2(n)$ splits** for list with n entries

example: list with $n = 128$ entries

step	number of lists	length of each list
0	1	128
1	2	64
2	4	32
3	8	16
4	16	8
5	32	4
6	64	2
7	128	1

for $n = 128$ we need $7 = \log_2(128)$ steps

Complexity of MergeSort?

- we found **recursive formula** for runtime of MergeSort

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$$

- for $n = 1$ we do not split or merge, i.e. for **base case** we have

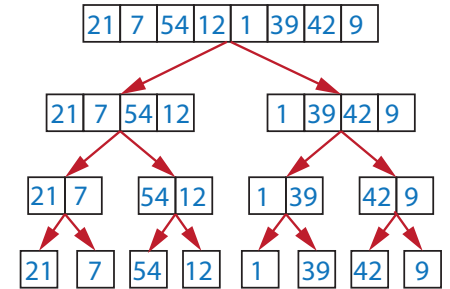
$$T(1) := 0$$

- we can now **recursively calculate ...**

- $T(2) = 2 \cdot T(1) + 2 = 2$
- $T(4) = 2 \cdot T(2) + 4 = 2 \cdot 2 + 4 = 8$
- $T(8) = 2 \cdot T(4) + 8 = 2 \cdot 8 + 8 = 24$
- $T(16) = 2 \cdot T(8) + 16 = 2 \cdot 24 + 16 = 64$
- $T(32) = 2 \cdot T(16) + 32 = 2 \cdot 64 + 32 = 160$

- one can **solve this recursive formula** as

$$T(n) = n \cdot \log_2(n)$$



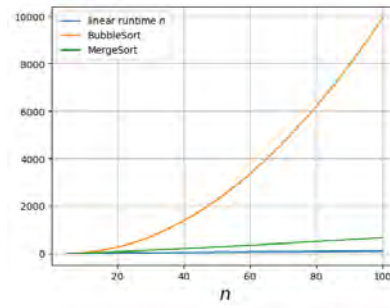
Notes:

Notes:

Complexity of sorting?

- ▶ with BubbleSort we can sort n numbers in $n - 1$ steps in best case and $n \cdot (n - 1)$ in worst case
- ▶ MergeSort improves worst-case complexity of BubbleSort from n^2 to $n \log_2(n)$
- ▶ **on average** MergeSort requires $n \log_2(n)$ steps
- ▶ to sort n objects based on **pairwise comparisons**, there is **no algorithm that requires less than $n \log_2(n)$ steps on average**
- ▶ but: there are specialized algorithms to **sort n integer numbers within a known range** with linear runtime

→ self-study questions



worst-case complexity of MergeSort vs. BubbleSort

Practice Session

- ▶ we implement BubbleSort in python
- ▶ we implement the **recursive** divide-and-conquer method MergeSort
- ▶ we investigate the **runtime of both algorithms** for increasingly large inputs

```
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]

        mergeSort(left)
        mergeSort(right)

        left = merge(left, right)

    return arr

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])

    return result
```

practice session

see directory 06-01 in **gitlab repository** at

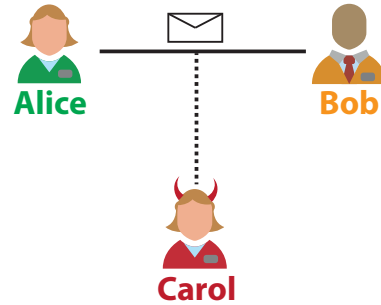
→ https://gitlab2.informatik.uni-wuerzburg.de/m4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

Cryptographic algorithms

- ▶ some of the **oldest algorithms in human history** are **cryptographic algorithms** that facilitate secure communication
- ▶ common scenario involves two persons **Alice (A)** and **Bob (B)** that wish to exchange messages via **public channel**
- ▶ **adversary Carol (C)** can intercept and/or manipulate messages
- ▶ how can Alice and Bob **securely communicate** despite adversary Carol?



Cryptography

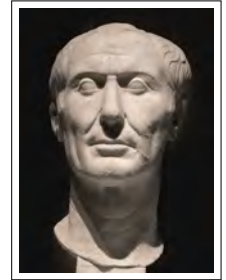
Cryptography refers to the practice and study of **secure communication** in the presence of **adversarial** attacks. → [Wikipedia](#)

A simple encryption algorithm

- ▶ we assume that Alice and Bob prearranged a **shared secret key** (e.g. during a previous secret meeting)
- ▶ Alice uses key to **encrypt** message in **plaintext** and sends **ciphertext** to Bob
- ▶ Bob uses key to **decrypt ciphertext** and obtains **plaintext**
- ▶ any idea for a **simple encryption algorithm**?

Caesar cipher

- ▶ **secret key** is number k between 0 and 25 (alphabet with 26 letters)
- ▶ each letter in plaintext is replaced by letter found by shifting letter in alphabet k positions to right
- ▶ positions larger than 25 are wrapped around to zero
- ▶ simple **substitution cipher** allegedly used by Julius Caesar to send military commands (using $k = 3$)



Julius Caesar
100 BC – 44 BC

image credit: Ángel M. Felicísimo, Wikimedia Commons, CC-BY-SA

0	1	2	3	4	5	6	7	...
a	b	c	d	e	f	g	h	...
d	e	f	g	h	i	j	k	...

Caesar cipher with key $k = 3$

Notes:

Notes:

Caesar cipher in python

- ▶ assume that plaintext message is given as **list of characters** in python, e.g.
['h', 'e', 'l', 'l', 'o', ' ', 'c', 'a', 'e', 's', 'a', 'r']
- ▶ simplest approach limited to 26 (lower or uppercase) letters in alphabet
- ▶ we can use Caesar cipher to shift **any character** based on its underlying binary encoding
- ▶ **use ASCII/Unicode table** to map characters to numbers and vice-versa, e.g. 'a' \leftrightarrow 97
- ▶ in python we can use functions `ord` (map character to unicode index) and `chr` (map unicode index to character)

plaintext:

hello caesar

cipher text:

khoor fdhvdu

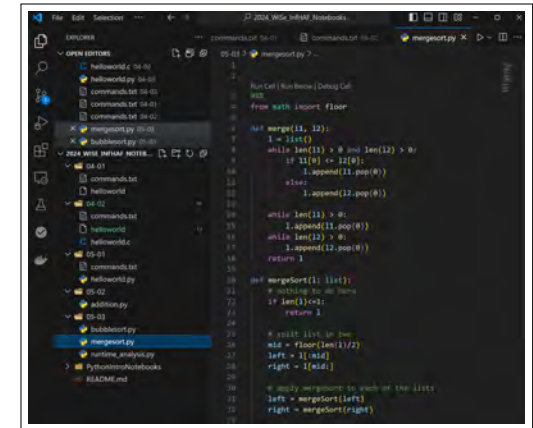
0	1	2	3	4	5	6	7	...
a	b	c	d	e	f	g	h	...
d	e	f	g	h	i	j	k	...

Caesar cipher with key $k = 3$

Notes:

Practice Session

- ▶ we implement the Caesar cipher in python
- ▶ we use it to encrypt and decrypt a message
- ▶ we investigate the **security of the Caesar cipher** and perform a **brute-force attack on the key**



practice session

see directory 06-02 in [gitlab repository](https://gitlab2.informatik.uni-wuerzburg.de/m4nets_notebooks/2024_wise_infhaf_notebooks) at

→ https://gitlab2.informatik.uni-wuerzburg.de/m4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

A (slightly) better approach

- ▶ how many different keys do we have in the Caesar cipher?
 - ▶ for alphabet with length $| = 26$ we have 26 possible shifts
 - ▶ for 16-bit unicode we have $2^{16} = 65536$ possible shifts
 - ▶ takes less than 1s on common computer
- ▶ Caesar cipher is extremely easy to break by **brute-force attack** that simply tries each possible key
- ▶ idea: instead of limiting ourselves to shifts, we could apply **arbitrary substitution of characters**
- ▶ secret key = **substitution table** that gives substitution for each character in plaintext
- ▶ how efficient is a **brute force attack** now?

a	b	c	d	e	f	g	h	...
j	t	x	f	z	a	f	k	...

substitution table

Security of substitution ciphers

- ▶ how many different **permutations** do we have for $['a', 'b', 'c']$?
- ▶ for alphabet with n letters we have
$$n! := n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 2 \cdot 1$$
possible substitutions
- ▶ factorial $n!$ can be calculated in **recursive fashion**
$$n! = n \cdot (n - 1)!$$
- ▶ for 26 letters and assuming we can test 1 billion keys per second, **brute-force attack may take 3.2 billion years**
- ▶ for $n = 65$, number $n!$ of possible keys is **larger than number of atoms in the universe**
- ▶ is this really secure?

n	$n!$
1	1
2	2
3	6
4	24
5	120
6	720
...	...
25	$1.55 \cdot 10^{25}$
26	$4.03 \cdot 10^{26}$

Notes:

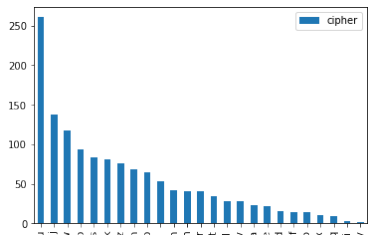
Notes:

- It is easy to see that each substitution is just a different sequence of the 26 letters of the alphabet.
- In order to understand in how many different ways we can substitute 26 letters of the alphabet, we thus have to consider so-called permutations, i.e. the possible ways in which we can arrange those letters.
- Let us consider this for an alphabet with three letters a, b, and c. For the first position, we have three choices (a, b, c). This choice can be combined with all choice for the second position, however since we already fixed the first one there are only two choices left. This gives a total of $3 \cdot 2$ for the first two positions. Fixing those two positions also fixes the last one (since we only have three letters), i.e. there is only one choice left and we thus have a total of $3 \cdot 2 \cdot 1 = 6$ possible ways in which we can arrange three letters.
- Using the same reasoning, for four letters we find $4 \cdot 3 \cdot 2 \cdot 1 = 24$ different permutations and for the general case of n letters we have $n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 1$.
- This number is called the **factorial $n!$** of n and it plays an important role in combinatorics. In the table above, you can see that the factorial grows extremely fast!
- Because of this fast growth, the number of possible keys is so large that a brute-force attack is not efficient. However, this still does not imply that this simple cipher is secure!

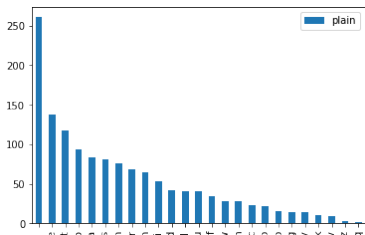
Crpytographic analysis

Carol receives the following cipher text encrypted using substitution method with unknown key

zxuyguxhuexuzxuyguzvczukluzvgusjglzkxeupvgzvghuzkluexynghukeuzvguokeiuzxuljwwghuzvgu
luceiuchhxpluxwuxjzhc gxjluwxhzjeguxhuzxuzcbgucholuc ckelzuculgcuxwuzhxyngluceiuytuxd
ugeiuzvgouzxiikguzxulnggduexuoxhguceiuytuculnggduzxlutpgug iuzvguvgchzucrvguceiuzvguzvx...



character frequencies in cipher text



character frequencies in English texts

apart from brute-force attacks, we can use cryptanalysis to reduce the space of possible keys

Notes:

- By simply calculating the frequencies of letters in the cipher text and comparing them to the frequency of characters in the (known) language of the plaintext, it is often possible to guess a sufficient number of substitutions to actually decrypt the message. In any case, this reduces the space of possible keys so that we can break the encryption!
- In our example, it is immediately clear that the character *t* is the substitution of the most frequent character *e*.

Vigenère cipher

- monoalphabetic substitution is easy to break using brute-force or frequency analysis
- idea: use polyalphabetic substitution that applies multiple substitutions for same character

example: key $k = 5237$

- plaintext message = "hello world"
- for first character apply Caesar shift with $k = 5$
- for second character apply Caesar shift with $k = 23$
- for third character apply Caesar shift with $k = 7$
- for fourth character apply Caesar shift with $k = 5 \dots$

k	a	b	c	d	e	f	g	h	i	...
0	a	b	c	d	e	f	g	h	i	...
1	b	c	d	e	f	g	h	i	j	...
2	c	d	e	f	g	h	i	j	k	...
3	d	e	f	g	h	i	j	k	l	...
4	e	f	g	h	i	j	k	l	m	...
5	f	g	h	i	j	k	l	m	n	...
6	g	h	i	j	k	l	m	n	o	...
7	h	i	j	k	l	m	n	o	p	...
8	i	j	k	l	m	n	o	p	q	...
9	j	k	l	m	n	o	p	q	r	...

Vigenère table

- proposed by Blaise de Vigenère in 1585
- unbreakable if key is as long as the message (so-called one-time pad)

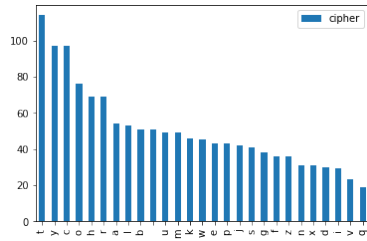
Notes:

- Their vulnerability to frequency analyses is a problem of all monoalphabetic substitution methods, i.e. methods that always replace one letter in the alphabet by another letter.
- We can solve this issue by polyalphabetic substitution methods that use a more sophisticated approach that replaces the same letter by different letters.
- The so-called Vigenere cipher is an example for such a method. It is based on a table, which lists all possible Caesar shifts. The idea is then to use a different shift for each character, depending on its position. Which shift is used for a given character is determined based on a key, which gives the sequence of shift values that are simply repeated if the message is longer than the key.
- You can see that the Caesar cipher is just a special case of the Vigenere cipher where the key contains only a single number (which is thus applied to all characters).
- In fact, whenever we repeat the key because the message is longer than the key, we use the same mapping again, which can again make the method vulnerable to frequency analysis if the message is much longer than the key. The same holds if we reuse the same key for multiple messages!
- However, if the key is as long as the message and we use the key only once, the Vigenere method is unbreakable (and it is thus still used today in the so-called one-time pad encryption).

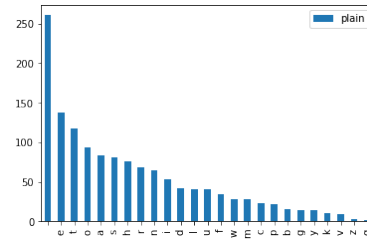
Frequency analysis?

Carol receives the following cipher text encrypted using Vigenère encryption with unknown key

whoveyyutbhtyrcrtqy rrdmobsyckyojucbwbcg urhmwyrycllogo vkhobnyckyofilncmtsspiyfttfocl bnebcubx
mpchmrzqhhl dyumigeyyuthh rknyourkbcuvuilbwtptsckchuttpyxv sykqxovyysicilqcybx ...



character frequencies in cipher text

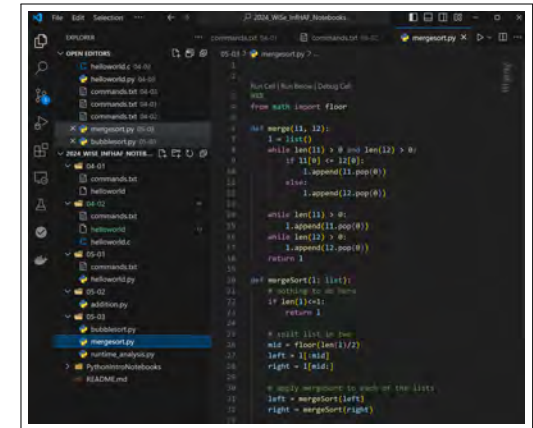


character frequencies in English texts

depending on key length, polyalphabetic substitution hinders application of cryptanalysis

Practice Session

- ▶ we implement the substitution and Vigenere cipher in python
- ▶ we use it to encrypt and decrypt a message



practice session

see directory 06-03 in gitlab repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/m4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

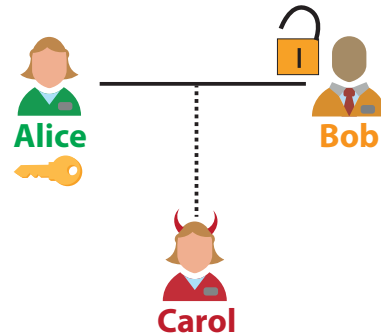
Notes:

Public-key cryptography

- ▶ so far we assumed that Alice and Bob use **symmetric encryption methods** that require a **shared secret key**
- ▶ is this practical (e.g. on the Internet)?
- ▶ **public key cryptography** refers to **asymmetric methods** that do not require shared secret key
- ▶ consider analogy where Alice and Bob use **physical key and lock** to securely communicate

idea

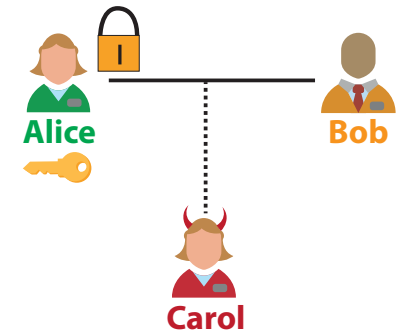
- ▶ Alice has access to locks and a corresponding **key**
 - ▶ Alice sends **open locks** to any communication partner
 - ▶ Bob **uses lock on a box**, and sends locked box with message to Alice
 - ▶ Alice **uses her key to open locked box** and reads message
-
- ▶ lock = **public key** used to “encrypt message”
 - ▶ key = **private key** used to “decrypt message”



A public-key encryption algorithm?

- ▶ public key uses **one-way function** that is easy to apply (i.e. close lock) but difficult to reverse (i.e. open lock)
- ▶ consider **prime factorization of a number**
- ▶ given prime numbers 79 and 37 it is **easy to compute product** $79 \cdot 37 = 2923$
- ▶ but: given number 2923 it is **difficult to find (unique) prime factors** 79 and 37
- ▶ basis for **RSA algorithm** that uses pair of prime numbers p, q as private key and $n = pq$ as public key

→ Rivest, Shamir and Adleman, 1978



Notes:

- importantly, different from shared keys of symmetric encryption algorithms the public key can be made available publicly (i.e. posting it on a website) as it can only be used to encrypt messages, not to decrypt messages
- anyone who wants to securely communicate with Alice, just obtains the public key and uses it to encrypt a message to her. Only Alice can decrypt these messages using her private key.

Notes:

Hard problems in computer science

- ▶ RSA algorithm relies on the fact that it is **difficult** to find prime factors of a large number
- ▶ but who can guarantee that no one will find an efficient algorithm (thus breaking RSA) in the future?
- ▶ prime factorization belongs to so-called **NP complexity class** of problems
- ▶ there are **no known algorithms** that can solve an NP problem faster than trying all possibilities (brute force)
- ▶ unclear whether we have not discovered such algorithms yet or whether they do not exist
- ▶ **algorithms for quantum computers** can efficiently solve some problems in NP and thus pose a threat for cryptographic methods → **post-quantum cryptography**



Homer in 3D

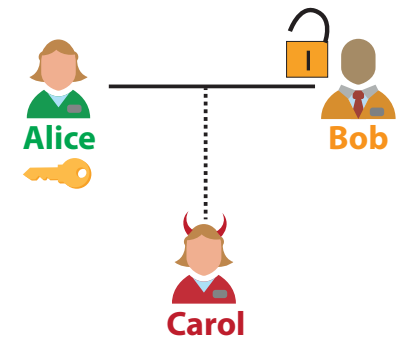
image credit: Screenshot from The Simpsons, Matt Groening & 20th Television, fair use

Digital signatures

- ▶ apart from encryption we often want to **verify the authenticity** of messages or data

use cases

- ▶ receiving an E-Mail from your friend
 - ▶ visiting the website of your bank
 - ▶ downloading a software from the Web
- ▶ how can Bob verify that received message is actually from Alice?
 - ▶ **idea:** use private/public keys in inverse fashion
 1. Alice uses her **private key to encrypt message**
 2. Bob uses Alice's **public key to decrypt message**
 3. successful decryption proves that message is actually from Alice
 - ▶ method/algorithm to verify authenticity of messages is called **digital signature**



Notes:

- We commonly refer to the class of problems that can be solved in polynomial time (e.g. quadratic, cubic, etc.) as P . In contrast, those problems for which the best we can do is to verify a known solution in polynomial time fall in the class of NP .
- Whether these two classes are actually identical (i.e. $P = NP$), that is whether we can solve problems in NP in polynomial time is the biggest open question in computer science. The consequences of $P = NP$ would be so drastic (i.e. we would suddenly be able to efficiently solve incredibly hard problems), that most computer scientists believe that the two classes are not equal.

Notes:

History of public-key cryptography

- ▶ until 1970s: government agencies (i.e. secret services) held monopoly on
- ▶ first publicly proposed public-key cryptographic algorithm was developed by **Diffie and Hellman** in 1976
- ▶ in 1991 Phil Zimmermann publicly released software **Pretty Good Privacy (PGP)**, which implemented RSA algorithm
- ▶ resulted in investigation by US Customs service for violation of **Arms Export Control Act**
- ▶ protocols using public-key cryptography have since become **key infrastructure for secure Internet communication** (e.g. SSL/TLS, S/MIME)

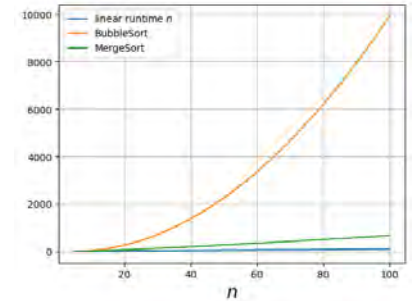


Phil Zimmermann
born 1954

image credit: User Beao, Wikimedia Commons, CC-SA

In summary

- ▶ we introduced two fundamental algorithms to address the **sorting problem**
- ▶ we showed how we can implement the divide-and-conquer algorithm **MergeSort** in a recursive fashion
- ▶ we introduced basic **symmetric encryption algorithms** and discussed their limitations
- ▶ we explained principles behind **asymmetric public-key cryptography** which can be used to **encrypt and authenticate** messages
- ▶ we highlighted that cryptographic methods often rely on **known hard problems in computer science**



Notes:

Notes:

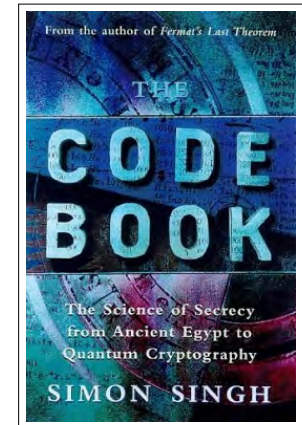
Self-study questions

1. Give a formulation of the BubbleSort algorithm in python and explain it in your own words.
2. Give an example for an input for which the BubbleSort algorithm performs the maximum/minimum number of comparisons.
3. Count the number of swaps in an input list with n elements, where BubbleSort performs the maximum number of comparisons.
4. Investigate the BucketSort algorithm for integers in a fixed range and explain why it takes less than $n \log_2 n$ steps on average.
5. Give a formulation of the MergeSort algorithm in python and explain it in your own words.
6. Explain why monoalphabetic substitution methods like the Caesar cipher are not secure.
7. Explain why we can - in general - not use frequency analyses to break the Vigenère cipher.
8. What is the difference between symmetric and asymmetric encryption algorithms?
9. Assuming you can test one billion keys per second, calculate how many years a brute-force attack can take to break a monoalphabetic substitution cipher with an alphabet of 26 characters.
10. Explain how we can use public-key cryptography to securely communicate without secretly exchanging a shared key.
11. Explain how public-key cryptography can be used to verify the authenticity of messages.

Further reading

References

- ▶ EH Friend: [Sorting on Electronic Computer Systems](#), Journal of the ACM, Vol. 3, 1956
- ▶ W Diffie, ME Hellman: [New Directions in Cryptography](#), IEEE Transactions on Information Theory, 1976
- ▶ R Rivest, A Shamir, L Adleman: [A Method for Obtaining Digital Signatures and Public-Key Cryptosystems](#), 1978
- ▶ Simon Singh: [The Code Book](#), Fourth Estate, 1999



Notes:

Notes:

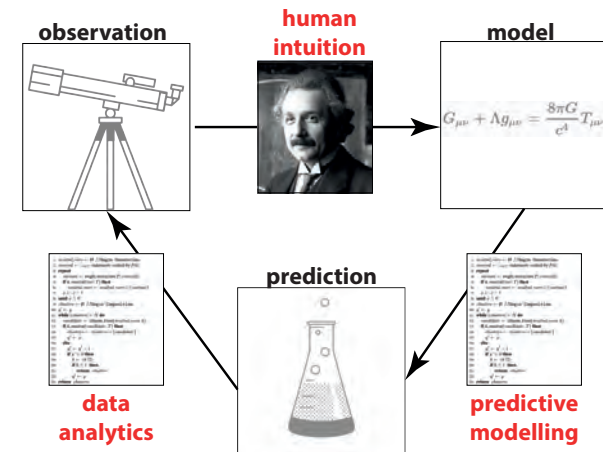
Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

What is learning?



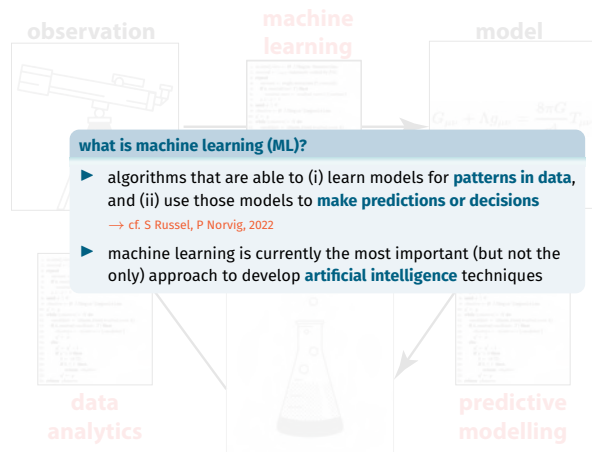
Notes:

- **Lecture L07: Machine Learning and AI I** 03.12.2024
- **Educational objective:** We introduce machine learning and explain how it relates to the scientific method. We explain the logistic regression model and apply it to an image recognition task.
 - What is Machine Learning?
 - Supervised Learning: Classification
 - Model Complexity and Overfitting
- **Exercise Sheet 5** 17.12.2024

Notes:

- Before we can understand what “machine learning” is, we first need to understand what “learning” is. Learning refers to the **systematic process by which we extract knowledge from experience** (cf. Greek “empeiria” for experience). This experience is based on the observation of a phenomenon in our environment. Classical examples are astronomical observations (e.g. of stars, planets, or galaxies).
- We are interested in a model that describes (or explains, from Latin “explano” for “make plain” or “make clear”) our observations. A **good model provides a compact summary of our observations and allows to derive predictions** that can be used to falsify our model in future observations.
- We call this systematic process the “scientific method” and digital technologies have revolutionized it, e.g. by improving our ability to derive predictions through large-scale simulations or test predictions in vast amounts of data. However, the task of deriving a good model that can be used to make predictions is largely left to human intuition and creativity.

What is machine learning



Demystifying machine learning

supervised learning
“learn” model in labeled data



exemplary algorithm

we introduce **logistic regression** to address binary **classification problems** in labeled data
→ now

unsupervised learning
detect patterns in unlabeled data



exemplary algorithms

we introduce **k-means clustering** to detect **cluster patterns** in unlabeled data → LOS

Notes:

- Human cognitive abilities are limited, which is why we are interested in **artificial intelligence** techniques that can be used to further automate knowledge generation.
- Fitting this view, the terms “machine learning” refers to methods, processes, and algorithms that can be used to extract knowledge from large volumes of structured and unstructured data.
- Importantly, not all artificial intelligence techniques are based on machine learning in large volumes of data. There are also logic-based techniques which use knowledge stored in a database to make inferences based on logical rules. This different approach is called symbolic AI.

Notes:

Optical Character Recognition



how can we write an algorithm that “reads” numbers?

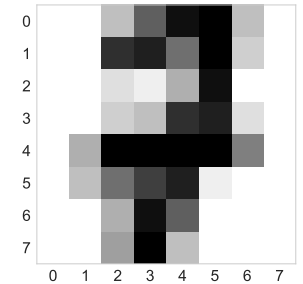
- ▶ **optical character recognition (OCR)** refers to the problem of **converting images of typed, handwritten or printed text into machine-encoded text**
- ▶ can be formulated as a **classification problem**

Classification problems

- ▶ consider an **8 x 8 pixel image** that shows exactly one digit
- ▶ for 8-bit greyscale images, **pixels capture 256 shades of grey** from 0 (white) to 255 (black)
- ▶ for given image, we want to develop an algorithm that “predicts” the digit shown in the image

classification problem

- ▶ **classification algorithm** takes some data x as input and outputs (or “predicts”) a **class label** c for data x from a set of k classes c_1, c_2, \dots, c_k
- ▶ for the special case of two classes (0 or 1, True or False, Positive or Negative) we call this a **binary classification problem**
- ▶ fundamental **supervised machine learning problem** with many practical applications



example

- ▶ input image with 8×8 grayscale values, i.e.

$$\underline{x} = (x_1, x_2, \dots, x_{64})$$
- ▶ output: different digits, i.e. classes

$$\underline{0}, \underline{1}, \underline{2}, \underline{3}, \dots, \underline{9}$$

Notes:

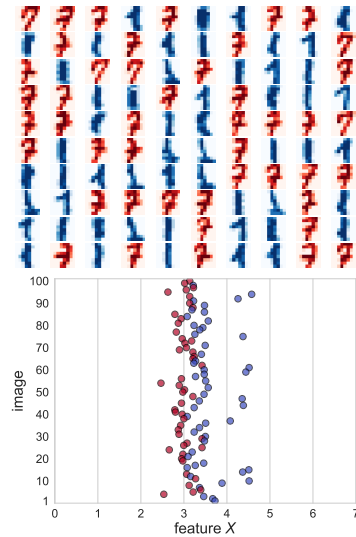
Notes:

Representation of data

- ▶ we often have **data with many dimensions**
 - ▶ 8 x 8 pixel image \Rightarrow 64 pixel values
 - ▶ reality: millions of pixels
- ▶ key challenge in ML is to find **low-dimensional representation** of data that preserves patterns needed to solve learning task
- ▶ can be done manually (**feature engineering**) or automatically (**representation learning**)

simple example for feature engineering

- ▶ for simplicity, we only consider **two digits “7” and “1”**
- ▶ compute **weighted average** of x coordinate of all pixel values, i.e. “center of mass” of images in horizontal dimension
- ▶ we **reduced dimensionality** of data from 64 to one dimension
- ▶ **pattern expressed in feature space** allows us to distinguish these two numbers

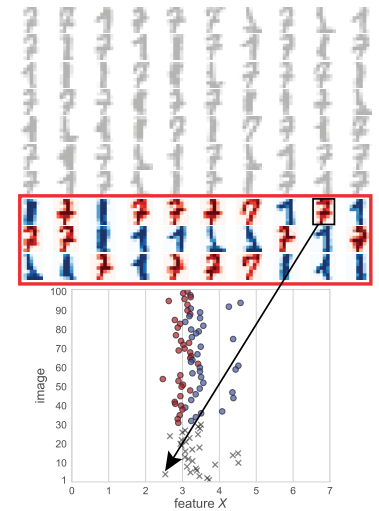


Training and cross-validation

- ▶ for **supervised classification** we use a **labelled training set** of images to **train a machine learning model**
- ▶ trained model should capture **patterns** that help us to classify future observations
- ▶ different from previously discussed algorithms, machine learning algorithms “**learn**” **patterns in training data**
- ▶ to evaluate our model, we **predict classes for test images** not used during training and compare predictions to **ground truth classes**

glossary: cross-validation

Cross-validation is an evaluation technique that splits data in a **training and a test set**. The training set is used to train the model, the test set is used to validate **how well the predictions of the model generalize** to independent data.



Notes:

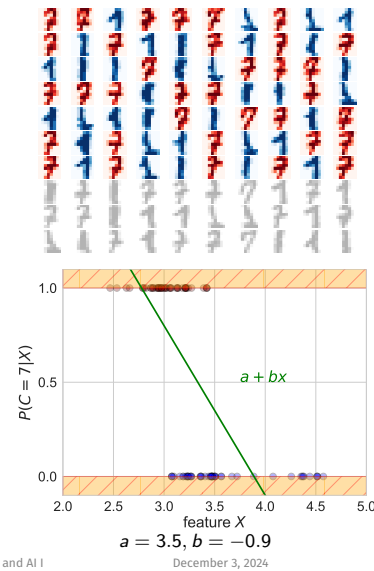
Notes:

A probabilistic classification method

- ▶ **idea:** use training data to compute **probability** $P(C = c|x)$ that image with feature x belongs to class c
- ▶ need a **model that describes how probability of a class depends on value of feature x**
- ▶ consider **linear model** that describes how the probability of one of the classes depends on feature x , i.e.

$$P(C = 7|x) = a + b \cdot x$$

- ▶ “model” = line defined by **two parameters a and b** that we want to “learn” in our training data
- ▶ **problem:** line $a + b \cdot x$ may not be between 0 and 1, i.e. **cannot interpret model output as probability**

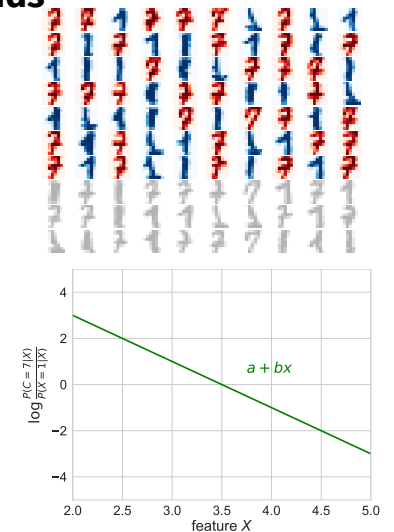


From probabilities to logarithm of odds

- ▶ instead of class probability, consider **odds**, i.e. how much more likely is **class $C = 7$** than **class $C = 1$**
- $$\frac{P(C = 7|x)}{P(C = 1|x)} \in [0, \infty]$$
- ▶ since probabilities range between 0 and 1, odds ranges between 0 and ∞ , i.e. still does not match linear model
- ▶ for number n between 0 and ∞ (**natural**) **logarithm** $\ln(n)$ of n ranges between $-\infty$ and ∞ , i.e.

$$\ln \frac{P(C = 7|x)}{P(C = 1|x)} \in [-\infty, \infty]$$

- ▶ we can use linear model to describe relationship between logarithm of odds (log-odds) and x



Notes:

- A very simple model for the relationship between feature x and class probability is a linear model, which assumes that the class probability proportional increases or decreases as we increase the feature value.
- From your basic math classes in high-school you may remember that a line $y = a + b \cdot x$ is defined by two parameters.
 - the intercept (a) defines the point at which the line crosses the y -axis. This is easy to see if you set $x = 0$, in which case we just have $y = a$.
 - the slope (b) controls how steep the line is and whether it runs from bottom left to top right (positive slope) or from top left to bottom right (negative slope). If we set the intercept $a = 0$ and $b = 1$ then we get a line $y = x$, which runs upwards from zero at a 45 degree angle.

Notes:

- In order to be able to apply the linear model to our problem, we need to find a different quantity to model, which actually matches the range $-\infty$ to ∞ of our linear model. We have seen that this is not the case for a probability that ranges between 0 and 1.
- We first consider the odds, which is simply the ratio between the probability of two possible outcomes. You might know from this betting, where odds are often used to indicate how much more likely one outcome is compared to another one (e.g. as in the odds for a horse winning are race are 1 to 5 or 1:5).
- Since probabilities range between zero and one, the odds ratio ranges between zero and infinity. A large value indicates that the probability of class 7 is much higher than the probability of class 1. A value close to zero indicates that the probability of class 7 is close to zero and the probability of class 1 is close to one.
- The problem is that the odds ratio still does not match the value range of our linear model, which ranges between $-\infty$ and ∞ . But there is a simple trick. By taking the logarithm, we can also map a number between 0 and ∞ to a number between $-\infty$ and ∞ . This is because the logarithm of one is always zero (since any number taken to the power of zero is one). Numbers $x < 1$ thus have negative logarithms, where the logarithm approaches $-\infty$ as x approaches zero.
- This implies that we can use a simple line to model the relationship between the logarithm of odds and the feature!

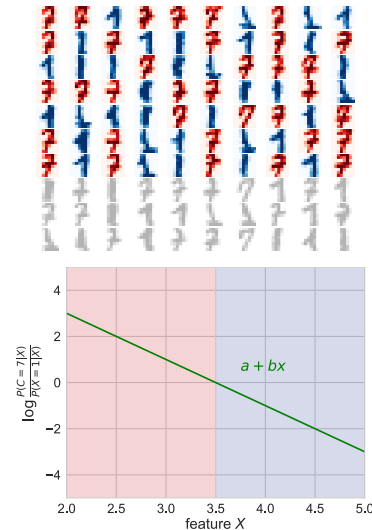
Decision boundary

- ▶ assume we have a line $a + b \cdot x$ that describes relationship between feature x and log-odds
- ▶ for any image with feature x we use output $a + b \cdot x$ of our model as **“prediction” of the log-odds**
- ▶ how can we use this to **assign a class** to an image?
- ▶ we have the following:

$$\frac{P(C = 7|x)}{P(C = 1|x)} > 1 \text{ when } P(C = 7|x) > P(C = 1|x)$$

$$\ln \frac{P(C = 7|x)}{P(C = 1|x)} > 0 \text{ when } P(C = 7|x) > P(C = 1|x)$$

- ▶ point x where line $a + bx$ crosses zero defines the **decision boundary between class 7 and class 1**



Logistic regression

- ▶ we can use **linear model** to describe how **logarithm of odds** for class 7 over class 1 depends on feature x , i.e.

$$\ln \frac{P(C = 7|x)}{P(C = 1|x)} = a + b \cdot x$$

- ▶ taking the exponent on both sides we get

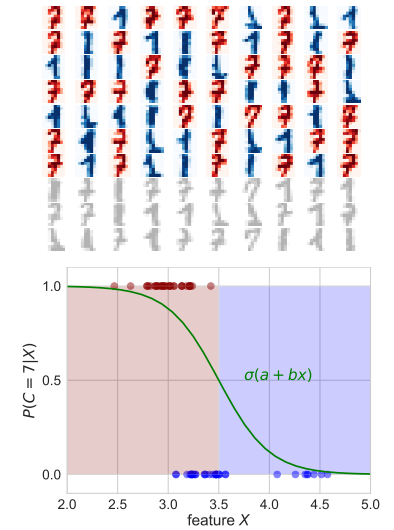
$$\frac{P(C = 7|x)}{P(C = 1|x)} = \frac{P(C = 7|x)}{1 - P(C = 7|x)} = e^{a+b \cdot x}$$

- ▶ we can calculate the **probability of class 7** as

$$P(C = 7|x) = \frac{1}{1 + e^{-(a+b \cdot x)}} := \sigma(a + b \cdot x)$$

glossary: logistic regression

Logistic regression is a statistical classification algorithm that uses the **logistic function** $\sigma(n) := \frac{1}{1+e^{-n}}$ to predict the probability that an object with feature x belongs to class c



Notes:

- Eventually, we would like to use our model to make a prediction or decision, i.e. we have to decide whether an image with feature x falls into class 7 or class 1.
- Intuitively, we should assign the class for which the probability is higher. If we consider the odds-ratio, this naturally leads to a decision point x at which the odds ratio is larger than one, at which point the probability of class 7 is larger than the probability of class 1.
- Since the logarithm of one is zero, for the logarithm of the odds, this leads to a decision point where the log-odds is larger than zero.
- In summary, this means that we use a linear model (i.e. a line) for the log-odds of the probabilities. As shown in the figure on the right, the point in the feature space (x -axis) at which this line crosses the value of zero on the y -axis is the point at which we switch the prediction of classes (highlighted by the background color). In the example, left of that point we predict that the image shows a seven, right of that point we predict that the image shows a one.

Notes:

- So we use a linear model for the logarithm of the odds-ratio between the classes. But we started with the idea to model the class probabilities, i.e. can we recover the actual probability from that model?
- In fact, it is easy to see how the model is related to class probabilities (and this will also show why the method is called logistic regression).
- Starting from the model ion (top) we can just take the exponent to the base e (Euler's number, which is the base of the natural logarithm) on both sides. Since $e^{\ln x} = x$ this simply removes the logarithm on the left side of the equation. Also, since we only have two classes we have $P(C = 1|x) = 1 - P(C = 7|x)$, which allows us to remove the $P(C = 1|x)$ from the equation.
- This can now be solved by $P(C = 7|x)$ and we are left with the equation above.
- The logistic function $\sigma(x) := \frac{1}{1+e^{-x}}$ takes values x between $-\infty$ and ∞ and maps it to values between 0 and 1 that **can be interpreted as probabilities**

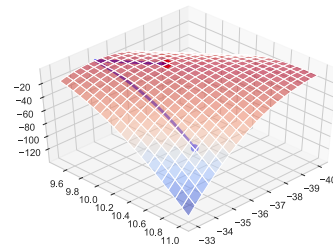
Loss function and model training

- ▶ for given parameters a and b logistic regression model gives $P(C = 7|x)$ for any feature value x
- ▶ for multiple training images (x_i, c_i) with features x_i and known classes c_i we can compute **quality of model** as

$$\mathcal{L}(a, b) = P(c_1|x_1) \cdot P(c_2|x_2) \cdot \dots \cdot P(c_n, x_n)$$

i.e. we **multiply probabilities of ground-truth classes** according to model

- ▶ larger values of $\mathcal{L}(a, b)$ mean that **model better fits our training data**
- ▶ to train our model, we must **find parameters a and b such that $\mathcal{L}(a, b)$ is maximal**



trajectory of gradient ascent algorithm

glossary: gradient ascent/descent algorithm

Gradient Ascent (or Descent) is an optimization algorithm that, starting from a random initial point in the parameter space, **moves step by step to find the maximal (or minimal) point of a function**. It is frequently used to train machine learning models.

Application to training images

let us apply **logistic regression** to a set of training images (using feature x with a single dimension)

1. **learn model parameters** in training data

gradient ascent algorithm

$$a = -37.5 \quad b = 10.2$$

2. use model to **predict class of image** based on feature x

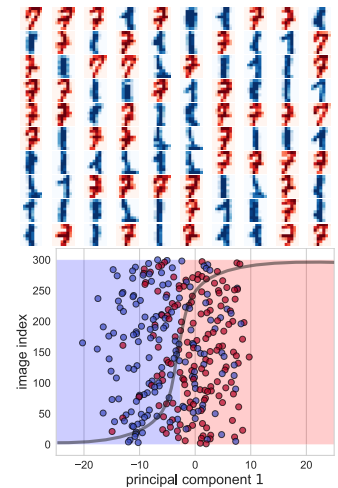
image 1

$$x = 3.78$$

$$P(C = 7|x) = 0.75$$

$$\Rightarrow C(x) = 7 \quad \checkmark$$

3. we obtain a **training error** of 33%



Notes:

Notes:

Going beyond one dimension

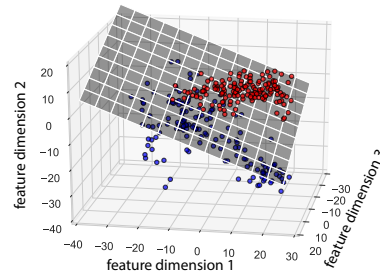
- ▶ we can extend logistic regression to **more than one feature**
- ▶ consider **linear model** that describes how probability of class depends on **two-dimensional feature** (x_1, x_2), i.e.

$$P(C = 7|\mathbf{x}) = a + b \cdot x_1 + c \cdot x_2$$

- ▶ for each additional feature dimension, we must learn one **additional model parameter**
- ▶ logistic regression model with any number of dimensions has **linear decision boundary**

linear decision boundary of logistic regression model

- ▶ one dimension: **decision boundary** is a **point**
- ▶ two dimensions: **decision boundary** is a **line**
- ▶ three dimensions: **decision boundary** is a **plane**



decision plane of a three-dimensional logistic regression classifier with parameters a , b , c and d

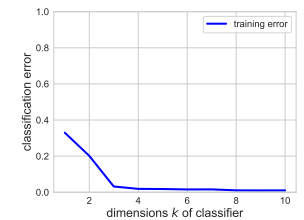
Increasing model complexity ...

- ▶ we fit logistic regression model for two classes based on **features with d dimensions**
- ▶ how does number of feature dimensions d affect **training error** of our model?

dimensions of our model	training error
1	0.33
2	0.19
3	0.03
≥ 8	0

glossary: underfitting

A one-dimensional logistic regression model **underfits** the pattern in the training data, i.e. we cannot distinguish classes with high accuracy in a one-dimensional feature space. We can often **reduce the training error** by increasing the dimensions of our model (i.e. using feature space with more dimensions).



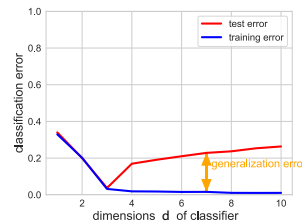
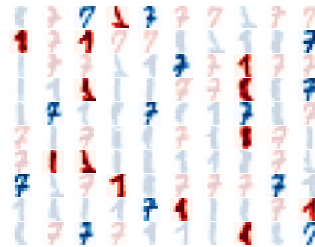
Notes:

Notes:

- For our example images, a one-dimensional logistic regression learns a decision boundary (point in a one-dimensional space) that correctly classifies approx. 67 % of all images, i.e. we have an error rate of approx. 33 %.
- For a two-dimensional classifier (now the decision boundary is a separating line in a two-dimensional space), we find that the error rate in our training data set drops to 19 %.
- For a three-dimensional logistic regression (which learns a separating plane in a three-dimensional feature space) the error rate drops to 3 %.
- We can obtain an error rate of zero if we further increase the number of dimensions beyond 8. We thus find that we need a sufficiently large number of dimensions to successfully capture the pattern in the feature space, i.e. to avoid an underfitting of our model.

Generalisation to new data?

- ▶ a good machine learning model is able to **generalise to new data** that were not used to train the model
- ▶ to evaluate this, we apply trained model to **test set** of images **not used during training** (cross-validation)
- ▶ how does number of dimensions affect the ability of our model to generalise?
 - ▶ for two dimensions, training and test error are approx. equal
 - ▶ for three or more dimensions, test error is larger than training error

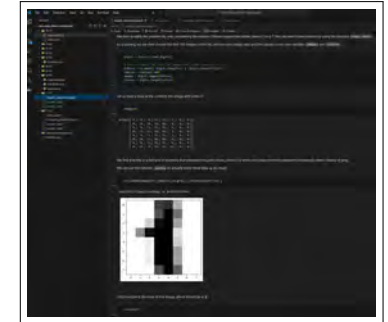


glossary: overfitting

Increasing the complexity of a machine learning model (e.g. in terms of number of parameters) can **reduce its generalisability** to unseen data. We call the process of tuning such a complex model to specific patterns that only exist in the training data **overfitting**.

Practice Session

- ▶ we show how to **train a logistic regression model** based on the machine learning library sklearn
- ▶ we apply logistic regression to a **data set of hand-written digits**
- ▶ we perform a **cross-validation** where we apply our trained model to a test data set



practice session

see directory 07-01 in **gitlab repository** at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

- This might lead us to think that we have solved our classification problem but it is not as easy as that. As explained in the introduction, **we are interested in generalizable models** that not only explain patterns in our observation but that make predictions about unseen data.
- We can assess this by means of a cross-validation, i.e. we apply our trained model to a test set of images that was not used in the learning phase.

Notes:

Evaluating binary classifiers

- ▶ we know how to **train a binary classifier** but how do we evaluate it?
- ▶ many decision problems can be formulated as **binary classification problem**
 - ▶ spam filters: SPAM (1) or NO SPAM (0)
 - ▶ medical diagnostics: DISEASE (1) or NO DISEASE (0)
 - ▶ fraud detection: FRAUD (1) or NO FRAUD (0)
 - ▶ credit scoring: CREDIT (1) or NO CREDIT (0)
 - ▶ jury decision: GUILTY (1) or NOT GUILTY (0)
 - ▶ ...
- ▶ classes are often interpreted as **positive (P, TRUE, 1)** or **negative (N, FALSE, 0)**
- ▶ **performance of a binary classifier** can be captured in a **confusion matrix**

		true class	
		neg.	pos.
predicted class	neg.	true negative <i>TN</i>	false negative <i>FN</i>
	pos.	false positive <i>FP</i>	true positive <i>TP</i>
		$N := TN + FP$	$P := TP + FN$

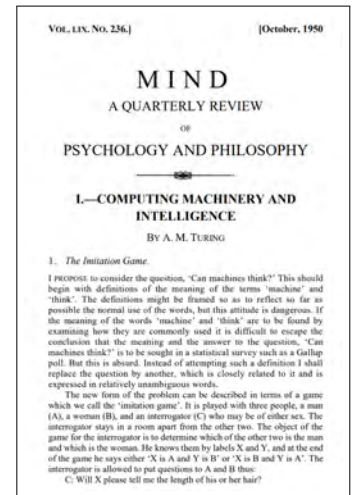
representation of **possible outcomes of a binary classifier** in a **confusion matrix**

classification errors are captured in **off-diagonal entries**

In summary

- ▶ machine learning = special class of **algorithms able to learn patterns in data**
- ▶ **supervised machine learning models** can be trained in labeled data
- ▶ **logistic regression** = supervised technique to predict classes based on features
- ▶ some models provide a **measure of confidence** in their predictions

machine learning \approx data + statistics + optimization algorithms
machine learning \neq intelligence



Notes:

- A number of interesting problems in AI can be framed as binary classification problems, e.g. whether a credit card transaction is legit or not, whether an E-Mail is spam, whether a blood sample indicates a disease, or whether the output of a production process is OK or faulty. How do we evaluate the accuracy of a binary classifier, e.g. for test data in the context of a cross-validation?
- We can put the four possible outcomes in a 2×2 confusion matrix, where diagonal entries capture cases where our class predictions are correct. The off-diagonal elements capture type I and type II errors, i.e. a false positive or a false negative (where we map dichotomous classes to positive/1 and negative/0).
- We can take the column sums of the matrix above to get the total number of positive and negative instances in our data. The relation between those two numbers gives an indication whether we have a **balanced** or **imbalanced** classification problem, i.e. whether one class is much more frequent than another class.
- Next week, we will see that we can generalise the confusion matrix to multiclass classification problems. Moreover, in the exercise sheet you will see that we can use the confusion matrix to calculate evaluation scores for binary classifiers.

Notes:

Self-study questions

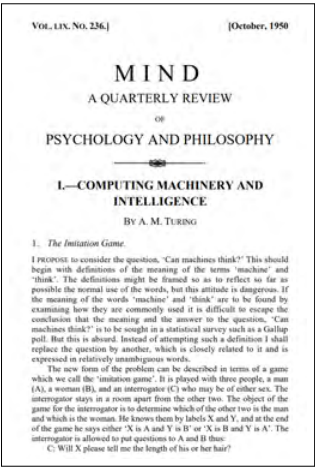
- 1. What is the difference between supervised and unsupervised learning?
- 2. Why is it useful to reduce the dimensionality of data before applying machine learning?
- 3. For which type of problems can we use logistic regression?
- 4. Why can we not simply use a linear model for the probability of a class, i.e. why can we not use the model $P(C|x) = a + b \cdot x$?
- 5. Explain how we can use the logistic function to calculate the class probability $P(C|x)$ based on the feature x in logistic regression.
- 6. Explain the difference between feature engineering and representation learning.
- 7. Consider a logistic regression model for a classification problem with two features x_1 and x_2 . What are the parameters of this model that we have to learn in the training data?
- 8. Explain how the decision boundary of a logistic regression model depends on the dimensionality of the feature space.
- 9. What is a classification problem? Give three examples.
- 10. Explain how the training and test set are used in supervised machine learning?
- 11. What is overfitting? Explain the term in the context of our logistic regression example.
- 12. What is underfitting? Explain the term in the context of our logistic regression example.

Notes:

Literature

reading list

- ▶ A Turing: **Computing Machinery and Intelligence**, 1950
- ▶ J Lighthill: **Artificial Intelligence: A General Survey**, 1973
- ▶ J Naisbitt: **Megatrends**, 1982
- ▶ KP Murphy: **Machine Learning: A Probabilistic Perspective**, MIT Press, 2012
- ▶ A Thamm et al.: **The Ultimate Data and AI Guide: 150 FAQs About Artificial Intelligence, Machine Learning and Data**, 2020
- ▶ S Russel, P Norvig: **Artificial Intelligence: A Modern Approach**, Pearson, 2022



Notes:

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

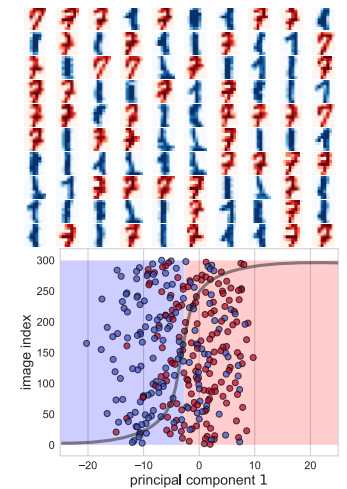
ingo.scholtes@uni-wuerzburg.de

Motivation

- ▶ we introduced **supervised and unsupervised machine learning**
- ▶ class of algorithms that **learn models for patterns in data**
- ▶ we considered **supervised classification problem**
- ▶ we showed how we can address it using **logistic regression**

open questions

- ▶ how can we **evaluate (binary) classifiers**?
- ▶ how can we learn in an **unsupervised fashion**
- ▶ what are **deep neural networks**?
- ▶ what is **generative AI**?



Notes:

- **Lecture L08: Machine Learning and AI II** 17.12.2024
- **Educational objective:** We continue our exploration of machine learning and AI. We first discuss how we can evaluate the quality of predictions by a (binary) classifier and introduce an unsupervised algorithm for cluster detection. We then explain artificial neural networks and discuss some recent developments in generative AI.
 - Evaluating the Quality of Classifiers
 - Unsupervised Machine Learning: Cluster Detection
 - From Logistic Regression to Artificial Neural Networks
 - Generative AI for Text, Images, and Videos
- **Exercise Sheet 6** 08.01.2025

Notes:

- In the previous lecture L07, we gave an overview of supervised and unsupervised machine learning algorithms, which are able to learn models for patterns in data.
- We specifically addressed a supervised classification problem (optical character recognition) and showed how we can address it based on an algorithm that is called **logistic regression**. Compared to previously covered algorithms that are tailored for a specific problem, e.g. to sort numbers, this algorithm works in a very different way. We use a training data set to “teach” it how to address our specific problem (e.g. detecting digits in an image). In logistic regression this simply works by finding those parameters of the model that best explain the pattern in the training data set (i.e. which points fall into one class as opposed to another class). Specifically, we have seen that - depending on the number of feature dimensions - with logistic regression we can learn a **linear decision boundary** in the feature space, which captures at which point one class becomes more likely than another class. In a one-dimensional feature space this decision boundary is a point, in two dimensions, it is a line, and in three dimensions it is a plane.
- What we have not addressed so far is how we can measure the quality of such a model, which is an important topic that - thanks to the EU AI Act - now also has a direct legal relevance. Interestingly for many classification problems evaluating the model is not as easy as just counting the number of wrong or correct predictions.

Evaluating binary classifiers

- ▶ we know how to **train a binary classifier** but how do we evaluate it?
- ▶ many decision problems can be formulated as **binary classification problem**
 - ▶ spam filters: SPAM (1) or NO SPAM (0)
 - ▶ medical diagnostics: DISEASE (1) or NO DISEASE (0)
 - ▶ fraud detection: FRAUD (1) or NO FRAUD (0)
 - ▶ credit scoring: CREDIT (1) or NO CREDIT (0)
 - ▶ jury decision: GUILTY (1) or NOT GUILTY (0)
 - ▶ ...
- ▶ classes are often interpreted as **positive (P, TRUE, 1)** and **negative (N, FALSE, 0)**
- ▶ **performance of a binary classifier** can be captured in a **confusion matrix**

		true class	
		neg.	pos.
predicted class	neg.	true negative <i>TN</i>	false negative <i>FN</i>
	pos.	false positive <i>FP</i>	true positive <i>TP</i>
		$N := TN + FP$	$P := TP + FN$

representation of **possible outcomes of a binary classifier** in a **confusion matrix**

classification errors are captured in **off-diagonal entries**

Accuracy and specificity

- ▶ consider the following confusion matrix of a binary classifier classifying 200 objects
- ▶ **accuracy** is probability that a prediction (positive or negative) is correct, i.e. fraction of correct predictions

$$ACC = \frac{TP + TN}{P + N}$$

- ▶ **specificity** or **true negative rate** is probability that the class of negative instances is correctly predicted

$$SPC = TNR = \frac{TN}{N}$$

		true class	
		neg.	pos.
predicted class	neg.	<i>TN</i> = 101	<i>FN</i> = 10
	pos.	<i>FP</i> = 15	<i>TP</i> = 74
		$N = 116$	$P = 84$

$$ACC = \frac{74 + 101}{200} = 0.875$$

$$SPC = \frac{101}{116} \approx 0.871$$

example

a diagnostic method that reliably rules out those patients that do not have a disease (true negatives) has **high specificity**, even if it misses many sick patients (false negatives)

Notes:

- The complexity of evaluating the quality of a classifier becomes obvious for binary classification problems, where an assignment to one of the two classes (0 and 1, yes and no, true or false) often has specific meaning and different implications.
- A number of interesting problems in AI can be framed as binary classification problems, e.g. whether a credit card transaction is legit or not, whether an E-Mail is spam, whether a blood sample indicates a disease, or whether the output of a production process is OK or faulty. How do we evaluate the accuracy of a binary classifier in a test data set where we know the ground-truth classes (e.g. during cross-evaluation).
- Since for each instance we can have two ground-truth classes and two possible predicted classes, there are four possible outcomes, where for two outcomes the classifier is correct and for two outcomes it is wrong. We can represent these four possible outcomes in a 2×2 confusion matrix, where diagonal entries capture cases where class predictions are correct. The off-diagonal elements capture so-called **type I and type II errors**, i.e. a **false positive** (negative class is wrongly predicted as positive) or a **false negative** (positive class is wrongly predicted as negative).
- We can take the column sums of the matrix above to get the total number of positive and negative instances in our data. The relation between those two numbers gives an indication whether we have a **balanced** or **imbalanced** classification problem, i.e. whether one class is much more frequent than another class.
- We will later see that we can generalise the confusion matrix to multiclass classification problems.

Notes:

- Using the entries in the confusion matrix, we can now define metrics that capture different aspects of a classifier's performance.
- The **accuracy is the probability that a prediction/classification is correct**, irrespective of the predicted class. It is simply the sum of the diagonal entries in the confusion matrix, divided by the total number of instances that are classified. A problem of this metric is that it does not distinguish between positive and negative predictions, which can be problematic if (i) we care more about one class than the other one, or (ii) if classes are imbalanced, i.e. there are many more negative than positive cases.
- The **specificity is defined as the probability that a negative instance is correctly classified**. This metric is useful in situations where false positives are particularly harmful (e.g. in spam detection, assuming that a positive class means spam). A high specificity implies that we have few false positives (i.e. few legitimate emails are wrongly classified as spam). A classifier that simply always assigns a negative class will have the maximum specificity of one (since there are no false positives). But such a classifier can still have a lot of false negatives!
- We generally favour models with high specificity in scenarios where a false positive has severe consequences (e.g. a false accusation/conviction in criminal investigations or court decisions). In a legal setting you can see the rule "in dubio pro reo" as a rule to ensure that court decisions have high specificity.

Precision, recall and f -score

- **sensitivity** or **recall** is probability that class of positive instance is correctly predicted

$$REC = TPR = \frac{TP}{P}$$

- **precision** is probability that a positive class prediction is correct

$$PRC = \frac{TP}{TP + FP}$$

- **F_1 -score** is the (harmonic) mean of precision and recall

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

		true class	
		neg.	pos.
predicted class	neg.	$TN = 101$	$FN = 10$
	pos.	$FP = 15$	$TP = 74$
		$N = 116$	$P = 84$

$$REC = \frac{74}{84} = 0.881$$

$$PRC = \frac{74}{74 + 15} \approx 0.831$$

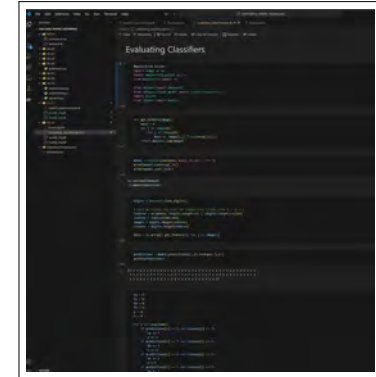
$$F_1 = \frac{148}{148 + 15 + 10} \approx 0.855$$

example

a diagnostic method that reliably detects patients that actually have a disease (true positives) has **high sensitivity/recall**, even if it wrongly diagnoses many healthy patients (false positives)

Practice Session

- we use the **confusion matrix and associated metrics** to evaluate our image classification approach
- we show how we can extend our classifier to **predict all 10 digits with very high accuracy**
- we show how we can use logistic regression to **quantify the uncertainty of the model** in the classification



practice session

see directory 08-01 in **gitlab** repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

- The **recall or sensitivity** is similar to specificity, but emphasizes false negatives rather than false positives. It is the fraction of positive instances that are correctly predicted. Again, a classifier that simply always assigns the positive class will have perfect recall or sensitivity because it can never produce false negatives. But such a classifier also has a low specificity (unless there are no or few negative instances).
- We should favour high recall in scenarios where a false negative has severe consequences and where we can accept false positives (e.g. in some medical diagnostic or quality assurance scenarios), where not detecting a disease or a faulty product (false negative) has severe consequences.
- **Precision captures the probability that an instance that is classified as positive actually belongs into the positive class.** Intuitively, it is easier to reach high precision if we predict less positive instances, i.e. classifying only those as positive where we have high levels of confidence. However, such a classifier would probably have a low recall/sensitivity.
- The overall accuracy of a classifier both in terms of false negatives and false positives can be measured in terms of the F_1 score, which is simply the (harmonic) mean of the precision and recall score.

Notes:

From supervised to unsupervised learning

supervised learning

“learn” model in labeled data



exemplary algorithm

we introduce **logistic regression** to address binary **classification problems** in labeled data
→ L07

unsupervised learning

detect patterns in unlabeled data



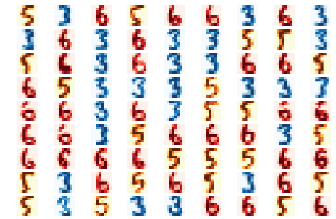
exemplary algorithms

we introduce **k-means clustering** to detect **cluster patterns** in unlabeled data → now

Cluster detection problem

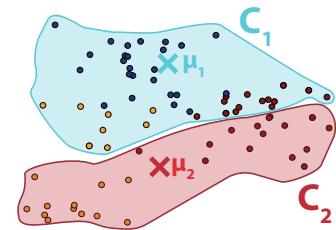
cluster detection problem

- ▶ split a set of data points x_1, \dots, x_n into k “natural” clusters such that **points in the same cluster are “similar” to each other**
- ▶ cluster detection is a fundamental problem in **unsupervised machine learning**
- ▶ OCR example: detect **groups of images that show the same digit**



- ▶ how can we evaluate how “similar” or “dissimilar” points in a given cluster are?
- ▶ idea: compute **distance of all points to “cluster center”**
- ▶ cluster center = **average of points in the cluster**, i.e. center μ_1 of cluster C_1 containing points x_1, \dots, x_m

$$\mu_1 := \frac{x_1 + \dots + x_m}{m}$$



Notes:

- Let us now reiterate on the distinction between supervised and unsupervised machine learning. So far, we have considered a supervised setting (e.g. classification) where we have access to a labeled data set that we can use to train a model (e.g. the logistic regression model).
- But what if we do not have labeled examples? Can we still detect patterns?

Notes:

- The cluster detection problem is an important example for an unsupervised machine learning problem that does not require labeled training data. For a set of objects (e.g. images that can again be represented by points in a two-dimensional feature space) we can try to split those objects into multiple groups or clusters such that objects in the same cluster are more similar to each other than objects that are placed in different clusters. In our OCR example, this can allow us to find clusters of images (in the feature space) that show the same digit, even if we do not know which digit is shown by any of the images!
- How can we calculate how “similar” or “dissimilar” the points in a given cluster are? We can just compute how far points in a given cluster are - on average - from the center of that cluster. In the extreme case that all points are identical (i.e. maximally similar) this would mean that they all fall onto the center, i.e. their “dissimilarity” would be zero. If points are distributed in a wide area, this means that the average distance to the center and thus their “dissimilarity” is large.
- We can easily compute the center of a cluster by taking the averages of all points, i.e. for two points (1, 1) and (2, 0) the cluster center is (1.5, 0.5).

Quality of cluster assignments

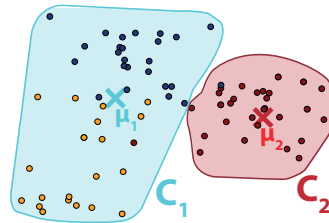
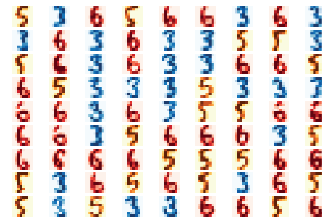
- ▶ consider two clusters C_1 and C_2 with **cluster centers** μ_1 and μ_2
- ▶ for cluster C_1 we can compute **distance of points within cluster C to center** as

$$L(C_1) = \text{dist}(x_1, \mu_1) + \text{dist}(x_2, \mu_1) + \dots$$

- ▶ total **dissimilarity of points within both clusters** is

$$L = L(C_1) + L(C_2)$$

- ▶ we can use L to compare **quality of different cluster assignments**, where smaller values of L indicate more “natural” clusters



$$L = 103.21$$

Group Exercise 08-01

- ▶ Assume that we want to detect $k = 2$ clusters in a data set with $n = 10$ objects. For each possible split of objects into clusters C_1 and C_2 , we calculate $L = L(C_1) + L(C_2)$ and choose the split with the smallest value L . For how many different splits do we need to calculate L ?

- ▶ We can represent cluster assignments as 0 and 1, i.e. each assignment of ten objects to the two clusters can be represented as a binary number with 10 digits.
- ▶ There are $2^{10} = 1024$ different binary numbers with 10 digits. For two of those (0000000000 and 1111111111) all points are assigned to a single cluster only, leaving $2^{10} - 2 = 1022$. If we replace 1 and 0 we actually get the same split (we just renamed the two clusters), i.e. we only need to consider half of these 1022 binary numbers.
- ▶ The total number is thus $\frac{2^{10}-2}{2}$

- ▶ How does your solution to the previous question change if we consider a split of 1000 objects in two clusters?

- ▶ We then have $\frac{2^{1000}-2}{2} \approx 10^{300}$ possible splits, which is more than there are atoms in the whole universe!

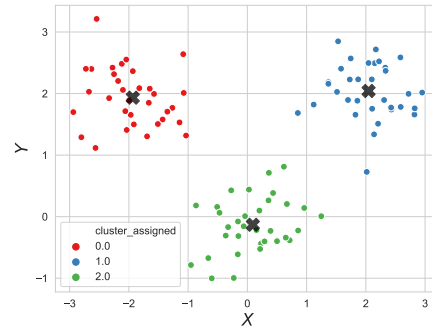
Notes:

- Now that we can compute how dissimilar points are within a given cluster, how can we compute the best possible split of points into a given number of k clusters? Let us assume that we have assigned points in some way to two clusters C_1 and C_2 . First, we can compute the dissimilarity of points in all clusters by simply adding the distances of all points in cluster C_1 to the cluster center μ_1 as well as the distances of all points in cluster C_2 to the cluster center μ_2 . This allows us to compute the total dissimilarity L of points within both clusters.
- This value of L tells us how well our assignment of points to the two clusters matches the natural cluster structure of the data, i.e. if we split the points in a more meaningful way we get smaller values of L . This also yields an idea for an algorithm: we simply have to find an assignment of points to clusters such that L is minimal.

Notes:

The k -means clustering algorithm

- ▶ how can we **optimally split points in k clusters** without trying each split?
- ▶ we can implement an **optimization algorithm** that quickly finds a good (but maybe not optimal) solution



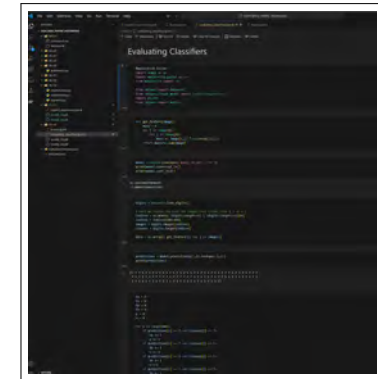
last step: update cluster centers

k -means clustering (Lloyd's algorithm)

1. **select k data points at random** and consider them as **center points** μ_1, \dots, μ_k of the k clusters
2. **assign all data points to clusters** that correspond to nearest center point
3. **update center points** of each cluster as the mean of points within that cluster
4. **repeat 2. and 3.** until clusters do not change anymore

Practice Session

- ▶ we implement the **k -means clustering algorithm** in python
- ▶ we test which clusters are found for different parameters k



practice session

see directory 08-02 in **gitlab** repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

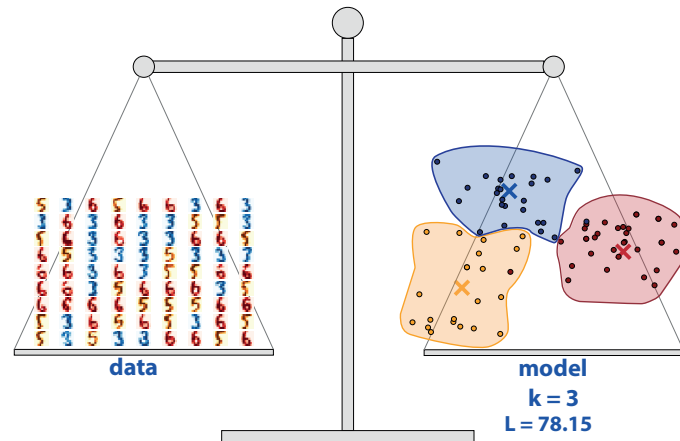
Notes:

- In theory, we compute the quality score L for all possible assignments of n points to k clusters, and then simply pick the one with the smallest value of L . Let us assume there are $n = 10$ points and we want to detect $k = 2$ clusters, which we can represent as zero and one. As seen in the previous group exercise, for $n = 1000$ points we then have $\frac{2^{1000} - 2}{2} \approx 10^{300}$ possible cluster assignments, a number that is (much!) larger than the number of atoms in the universe!
- This approach is clearly not feasible, which is why we need efficient (heuristic) algorithms that give us a good (but maybe not the optimal) solution. The k -means algorithm does that and it works as follows:
- We first select k of our points at random and simply consider those as the centers of our (initial) k clusters. Those k points can be naturally assigned to the corresponding clusters, but what about the other points?
- It is natural to assign each point to the cluster whose center is nearest to the point. However, now each of the k clusters contains more than just a single point so the center (which we just initialized based on the k points) will not be correct anymore.
- To correct for this, we can now recompute all cluster centers as explained before, i.e. we just take the mean of all points that are assigned to a given cluster.
- However, now that the cluster centers changed the assignment of points to clusters may be wrong, i.e. we should reassign all points to their nearest cluster center. This again changes the cluster center, and so on, i.e. we simply repeat those two steps until the clusters do not change anymore!

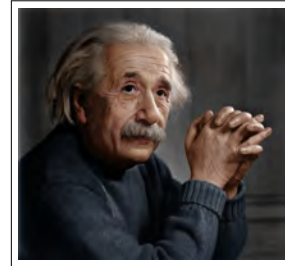
Notes:

- In the practice session we implement the k -means clustering algorithm and demonstrate it in a simple data set.

Choosing the number of clusters k ?



Over- and underfitting



fundamental challenge in AI

- ▶ complex models avoid underfitting, i.e. they are more likely to **explain the data well**
- ▶ simpler models avoid overfitting, i.e. they are more likely to **generalise to data not used during training**
- ▶ to avoid **overfitting and underfitting** we should use the **simplest model** that still **explains the data**

Notes:

- This brings us to an important issue in cluster detection: For the k -means algorithm, we have to specify the number of clusters k that we want to detect, but we often do not know this number in advance. How could we detect the optimal number of clusters k into which we can split our data?
- A simple (but wrong) idea could be to just run the k -means algorithm for different values of k and then choose the clustering where the value L is minimal. It is easy to see that this does not work if you consider how we can split a data set with n points into $k = n$ clusters. In the first step of our algorithm, we will simply assign one (random) point to each cluster. Since each point now has a distance to zero to its cluster center, this assignment is not changed anymore and the algorithm finished. Also, the cluster quality L is optimal because we have $L = 0$ (since each point is exactly at the center of its own cluster). So, if we run the algorithm for all values of k we will simply find a trivial solution where we assign each point to its own cluster.

Notes:

- This is another example of over- and underfitting of a model to the patterns in the data, similar what we have seen for logistic regression in the previous lecture.
- If we choose the k too small for the actual number of clusters in the data, we enforce a model that is too simple to explain the data, i.e. we underfit the patterns in the data. If we choose a value of k that is too large, we get a model that is unnecessarily complex and we will get cluster assignments that do not actually correspond to true cluster patterns, i.e. we overfit the pattern in the data.
- Just like for the classification problem, we need methods that allow us to balance between over- and underfitting.

What is a neuron?

- ▶ how does the **human brain** process information?
- ▶ human brain is a **network of approx. 85 billion nerve cells** (neurons)
- ▶ **neuron** is an electrochemically excitable nerve cell that consists of
 - ▶ tree trunk (soma) with multiple dendrites
 - ▶ nerve fiber (axon)
 - ▶ end feet (axon terminals)
- ▶ soma receives **synaptic inputs** via dendrites
- ▶ electrochemical input of dendrites **accumulates** in soma
- ▶ if inputs exceed a **threshold voltage** the neuron triggers **action potential** that travels across **axon**

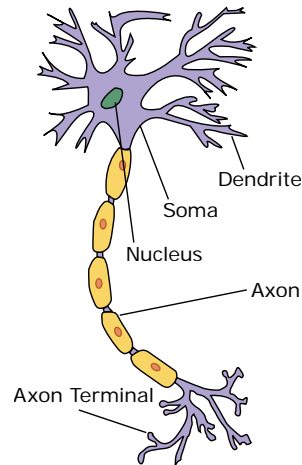


image credit: adapted from "Anatomy and Physiology", SEER Program

Notes:

- So far, we considered two simple machine learning algorithms, one for a supervised and one for an unsupervised problem. Let us now take a different perspective on the first algorithm (logistic regression) for classification, which will allow us to introduce (deep) neural networks.
- We first introduce the basic building block of neural networks: the perceptron model, a binary classifier that is based on a (grossly simplified model) for a single neuron in the brain.
- Why can we use a model for a neuron as a binary classifier? Let us have a look at the physiology of a neuron, an electrochemically excitable nerve cell that is depicted above. It features a number of branches (dendrites) that join into the tree trunk (soma). These dendrites have an electrochemical potential that accumulates in the soma to a certain threshold. If this threshold is exceeded, the neuron triggers a so-called action potential, which propagates across the nerve fiber (axons). This axon can be up to a meter in length and it terminates in multiple end feet (axon terminals), which can connect to the dendrites of other neurons.

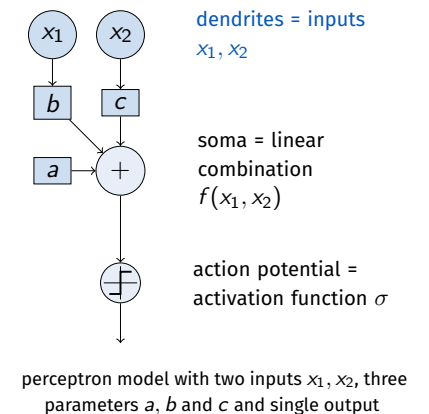
The perceptron classifier

- ▶ single neuron can be viewed as **binary classifier**
 - ▶ inputs = voltages received via dendrites
 - ▶ output = action potential (0 or 1)
- ▶ **perceptron** is function with inputs x_1, x_2, \dots, x_k and binary output $\{0, 1\}$ → F Rosenblatt, 1958
- ▶ soma = **linear combination** of inputs, i.e. for feature with two dimensions (x_1, x_2)

$$f(x_1, x_2) := a + b \cdot x_1 + c \cdot x_2$$
- ▶ **activation function** σ maps values between $-\infty$ and ∞ to values between 0 and 1, e.g.

$$\sigma(f(x_1, x_2)) = \frac{1}{1 + e^{-f(x_1, x_2)}}$$

where σ is **logistic function**

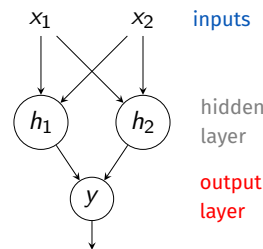
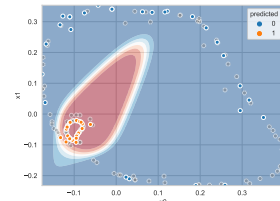


Notes:

- We can actually see each neuron as a basic binary classifier, which takes inputs (in the form of voltages) from the dendrites and generates a binary output (action potential or not) that serves as input for other neurons!
- Let us model this in a maximally simple form: We consider two inputs via dendrites via real values (features) x_1 and x_2 , which are multiplied by parameters b and c respectively. We further add a parameter a which models the "rest potential" of the neuron, i.e. how much voltage input it takes until the soma exceeds the threshold to "fire". The accumulation of inputs in the soma can be modelled by a linear combination of the parameter a and the inputs. An activation function models the triggering of an action potential, i.e. making it switch between zero and one as soon as a certain threshold value is exceeded.
- Where have we seen this before? Exactly, this is just the same as the logistic regression classifier, where we also computed a linear combination of the feature values with parameters a, b and c . Here, the logistic function served as **activation function** that allowed us to map the value $f(x_1, x_2)$ to binary classes 0 and 1 (or probabilities of class 1 in the range 0 and 1).
- This simple model of a binary classifier (which is equivalent to logistic regression) is also called the perceptron model, and it can be viewed as a simplified model for the behavior of a single neuron in the brain!

Artificial Neural Networks

- ▶ like logistic regression, perceptron has **linear decision boundary** (i.e. point in 1D, line in 2D, plane in 3D, etc.)
→ Exercise sheet
- ▶ we can connect multiple layers of perceptrons to an **artificial neural network**
 - ▶ first layer of perceptrons that take input features
 - ▶ outputs of one layer = inputs of perceptrons in next layer
 - ▶ output of last **output layer** are used as **predictions**
- ▶ layer of perceptrons that do not directly produce output is called **hidden layer**
- ▶ surprising fact: artificial neural network with single hidden layer can learn **decision boundaries with arbitrary shapes** → Universal Approximation Theorem



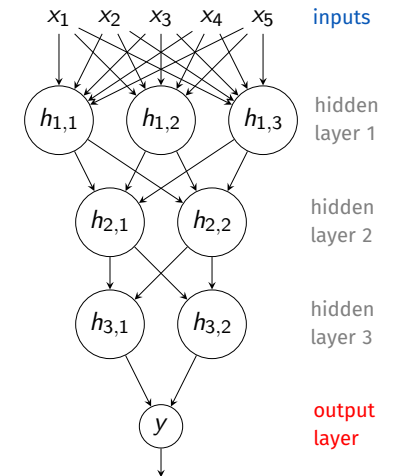
non-linear decision boundary of artificial neural network with a single hidden layer

Deep Neural Networks

- ▶ single perceptron/neuron with k inputs has $k + 1$ parameters
- example: artificial neural network with 1000 inputs and single output**

 - ▶ layer 1: 128 neurons = $128 \cdot (1000 + 1) = 128128$ parameters
 - ▶ layer 2: 64 neurons = $64 \cdot (128 + 1) = 8256$ parameters
 - ▶ layer 3: 16 neurons = $16 \cdot (64 + 1) = 1040$ parameters
 - ▶ output layer: single neuron = $1 \cdot (16 + 1) = 17$ parameters

total number of parameters = 137441
- ▶ to train artificial neural network we must find **optimal parameter values** for all perceptrons
 - ▶ **deep neural networks** can have **dozens of hidden layers** with billions or trillions of parameters
 - ▶ training such deep neural networks requires **clusters with fast GPUs**



neural network with eight neurons, **three hidden layers** and 35 parameters

Notes:

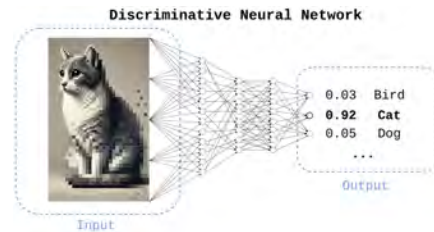
- Just like in logistic regression, the perceptron has a linear decision boundary, i.e. the boundary where we switch between two classes is a point if we have a single feature, a line if we have two features, a plane if we have three features and so on. Unfortunately, this is not enough to address many real classification problems where we often have more complex boundaries between classes (e.g. the circular boundary between the red and blue points above).
- To address this limitation, we can couple multiple layers of perceptrons and build an **artificial neural network**. Just like neurons in the brain, we can connect the output (axon terminal) of one neuron to the inputs (dendrites) of another neuron. A simple artificial neural network consisting of three perceptrons (i.e. neurons) is shown above. Here the first two neurons (called h_1 and h_2) both receive both input features x_1 and x_2 . They then both produce an output that is passed as an input to neuron y . The output of this neuron is then used to predict the class of our data!
- Perceptrons that do not directly produce an output that is used as a prediction (e.g. h_0 and h_1 in the example) are called **hidden neurons**. They form the **hidden layers** of the model. Neurons in the output layer take the output of hidden neurons and generate an output (y in the example above).
- While a single neuron (perceptron) always has a linear decision boundary, it might surprise you that it has been shown that an artificial neural network with a single hidden layer can actually learn arbitrarily shaped decision boundaries (like the circular one above).

Notes:

- How many parameters do we need to learn for such a neural network? For our simple example from the previous slide, we had two hidden neurons, each receiving two features. Each of those neurons has three parameters a , b , c , i.e. six in the hidden layer. Each hidden neuron produces one output, which is passed as an input to y , i.e. we have additional three parameters a , b , c in the output layer, yielding a total of nine parameters that we need to learn in our training data.
- The more neurons (and the more hidden layers) we have, the more powerful our artificial neural networks get, which is why we often use so-called **deep neural networks** with dozens or even hundreds of layers. For the example above, with eight neurons, three hidden layers and five input features, we get 35 parameters that we need to learn. For the example in the orange box (which is not particularly large) we already have more than 100,000 parameters!
- Real deep neural networks in current AI applications often have dozens of hidden layers, with billions or even trillions of parameters. As you can imagine, it is very resource-intensive to train such networks! It typically requires large compute clusters with thousands of very fast GPUs that can process thousands of parameters simultaneously.

From Discriminative to Generative AI

- ▶ we can use data to train **discriminative AI models** that **predict output** for a given input
 - ▶ image classification
 - ▶ optical character recognition
 - ▶ speech recognition
- ▶ need for (manually) labelled training data limits size of models that can be trained
- ▶ advances in deep learning since 2017 (e.g. “transformer” architecture) enable **self-supervised training** in massive data
- ▶ idea: mask part of input (text, image, etc.) and train neural network to predict missing part
- ▶ enables **generative AI models** that learn to generate new data

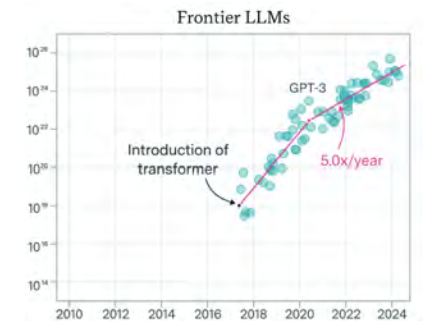


Notes:

- This brings us to generative AI, an important current application of deep neural networks. So far, we have only used machine learning in a discriminative way, i.e. we were interested to discriminate between different classes. Training such neural networks requires (manually) labeled training data, which limits the size of the data sets that we can use for training.
- Recent advances in deep learning (notably the so-called transformer architecture) have enabled a different **self-supervised** approach, where - instead of predicting manually assigned labels for data - we mask parts of the data and then train the model to predict missing parts in the input. This has enabled so-called generative AI models that - rather than assigning labels to data (e.g. assigning a text to an image) - actually learn to generate new data (e.g. generating an image from text)!
- This new approach has led to a recent surge in applications of AI in many different areas!

Large Language Models (LLM)

- ▶ LLM is a deep neural network that, given a sequence of input words (“prompt”) predicts a sequence of output words (“response”)
- ▶ basis of OpenAI’s ChatGPT is **Generative Pre-trained Transformer (GPT)** neural network architecture
- ▶ pretrained on **large volume of text collected from World Wide Web**
- ▶ fine-tuning of pretrained model based on **human feedback**
- ▶ GPT4 model rumored to have **1.7 trillion parameters**, requiring 100 mio USD of electricity for training
- ▶ LLMs have proven **useful to generate, summarize, or translate text**



number of floating point operations (Flops, y-axis) required to train different LLMs released over time (x-axis)

image credit: Epoch AI, Wikimedia Commons, CC-BY

Notes:

- A prominent example are so-called large language models (LLMs) that are trained to predict missing text. Given a sequence of input words (a so-called prompt) these models can then be used to predict a sequence of output words (the response to the prompt).
- A prominent example for such an LLM model is OpenAI’s ChatGPT, which uses a deep neural network architecture called Generative Pre-Trained Transformer (GPT). It has been trained on a huge (!) volume of text collected from the World Wide Web, which essentially covers almost all knowledge that humanity has produced (including all books, Wikipedia, billions of Web pages, news, etc.). This model is then further tuned based on human feedback, i.e. thousands of humans chat with ChatGPT and provide feedback which answers are useful and which are not.
- The GPT4 model behind ChatGPT is rumored to have more than 1 trillion parameters, and it takes approx. 100 mio USD in electricity alone to train this model (i.e. to find the optimal values of those parameters). The figure above shows the number of Floating Point Operations (i.e. addition, multiplication, division, etc.) that are required to train recent LLM models and we recently crossed the number of 10^{24} .
- Note that this very technical aspect of models is actually legally relevant since the recently released EU AI Act defines models that require more than 10^{25} Floating Point Operations as **General-Purpose AI models**, for which special regulations apply.

Generative AI for images, videos, and VR

- ▶ LLMs use text prompt as input and generate text as output
- ▶ given a text prompt, we can also use deep neural networks to **generate images or videos**
- ▶ **DALL-E and StableDiffusion** are **text-to-image models** that were trained on large volume of text and images from World Wide Web (released in 2021 and 2022)
- ▶ **SORA** is DALL-E based **text-to-video model** that was trained on large body of videos (release by OpenAI in December 2024)



Stable Diffusion result for prompt “a photograph of an astronaut riding a horse”

image credit: Wikipedia, User VulcanSphere

Notes:

- similar to LLMs that generate text for a text prompt, we can use text and images from the World Wide Web to train large models that generate images, video, or even whole 3D virtual reality scenes.
- Important recent models for image generation are DALL-E and StableDiffusion. Open AI just released its text-to-video model SORA, which allows to generate videos based on text prompts.
- As you can see the quality of these AI-generated images and videos is often remarkable!

Group Exercise 08-02

- ▶ Discuss the following video generated by the SORA text-to-video model.



Video generated by SORA model

image credit: Sora / OpenAI, public domain

- ▶ This video contains multiple problems, e.g. the transition of the arm into the bedsheet, or the anatomically incorrect paw of the cat.

Notes:

- ▶ Ask ChatGPT about the article “Ingo Scholtes: How to unify quantum theory and general relativity, Nature, 2024”. Discuss the answer.



image credit: OpenAI, ChatGPT, December 15 2024

- ▶ This article actually does not exist, the answer is a so-called hallucination!

Risks and Challenges in Generative AI

- ▶ current LLMs merely predict **response that is likely**, ignoring whether it is factually correct
- ▶ similarly, current text-to-image and text-to-video models do not consider **physical plausibility of output**
- ▶ AI-based generation of fake or implausible content is sometimes called **hallucination**
- ▶ malicious use of AI hallucinations to generate misleading content is called **deep fake**
- ▶ known weaknesses of generative AI can be used to **detect AI-generated deep fakes**
- ▶ research challenge: combine generative AI with **knowledge**



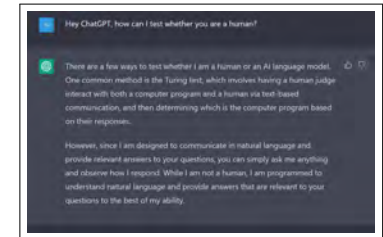
ChatGPT response to a prompt that asks to summarize a non-existing research article

image credit: OpenAI, ChatGPT, December 15 2024

AGI and Turing test

- ▶ “holy grail” of AI research is to develop a model that exhibits human-like **artificial general intelligence** (AGI)
- ▶ how can we **test whether a given algorithm exhibits “intelligence”**?
- ▶ problem: **no universally accepted definition of intelligence**
- ▶ Alan Turing proposed the following **“imitation game”**

I propose to consider the question, “Can machines think?”. This should begin with definitions of the meaning of the terms “machine” and “think.” [...] Instead of attempting such a definition I shall replace the question by another [...] The new form of the problem can be described in terms of a game which we call the **‘imitation game’**. [...] What will happen when a machine takes the part of A in this game? Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman? → A Turing, 1950



response from large language model
ChatGPT

image credit: OpenAI screenshot, taken on 29/03/2023

Notes:

- Now that you have a basic understanding for neural networks, we can also more easily understand their limitations. In fact, LLMs are not actually intelligent but rather predict which words are likely as a response to a text prompt - which they “learned” in a corpus of text that basically contains a large fraction of the text that humanity has produced in its history!
- However, finding a text that is likely as a response does not mean that it is factually correct. You can easily check this by asking ChatGPT to summarize an article that does not actually exist (see example above). We call this issue hallucination, as it somehow mimics dreams by the human brain.
- Similarly, generative AI models for images and videos only generate images and videos (or rather pixel patterns) that are likely even if they are not physically plausible. In the end, this is why we can use them to generate an image of an astronaut riding a horse on moon.
- The development of generative AI models that consider an underlying knowledgebase of facts or physical constraints is still an open research challenge that many researchers are currently working on.

Notes:

- Related to this, the holy grail of AI research is to develop a model that exhibits what we call human-like artificial general intelligence, i.e. a model that actually understands the underlying rules and laws and then uses them to reason.
- Unfortunately, this term is rather ill-defined as even for humans there is no universally accepted definition of intelligence (in fact, some current LLMs produce fairly good results in standardized intelligence tests but we would still not consider them to exhibit human-like intelligence).
- To address this, Alan Turing proposed a practical definition of intelligence that is based on the imitation game, i.e. whether during a remote correspondence a human is able to tell apart a human from an (intelligent) machine.

Conclusion

- ▶ we introduced **basics of machine learning and AI**
- ▶ we discussed **supervised vs. unsupervised problems** like **classification and cluster detection**
- ▶ we explored how we can **evaluate the prediction quality of classification algorithms**
- ▶ we explained basics of **artificial neural networks** that are the basis of **generative AI**
- ▶ we covered **open challenges and risks** of state-of-the-art generative AI models

I wish you merry christmas, peaceful holidays and a good start into a healthy and successful new year 2025!



DALL-E image for prompt: "Can you generate an image that is suitable to wish students of a university course a merry christmas and happy new year?"

image credit: DALL-E generated image
December 17, 2024

24

Self-study questions

1. How can we evaluate binary classification algorithms?
2. Give an example for a bad binary classifier that has a specificity of 1.0.
3. Give an example for a bad binary classifier that has a sensitivity of 1.0.
4. Give an example for a real-world classification problem, where high specificity is more important than high sensitivity.
5. Give an example for a real-world classification problem, where high sensitivity is more important than high specificity.
6. Explain why it is not enough to calculate either the precision or the recall of a classifier.
7. Explain the key idea behind Lloyd's cluster detection algorithm.
8. What is overfitting? Give an example based on the cluster detection problem?
9. What is underfitting? Give an example based on the cluster detection problem?
10. How is the perceptron model related to logistic regression?
11. How many parameters does a perceptron model with k inputs have?
12. How many outputs does a perceptron model have?
13. Explain the basic idea behind an artificial neural network. What is a hidden layer?
14. What is generative AI and what risks/problems does it introduce?

Ingo Scholtes

Introduction to Informatics

Lecture 08: Machine Learning and AI II

December 17, 2024

25

Notes:

Notes:

Literature

reading list

- ▶ A Turing: **Computing Machinery and Intelligence**, 1950
- ▶ M Minsky, S Papert: **Perceptrons: An Introduction to Computational Geometry**, 1969
- ▶ J Lighthill: **Artificial Intelligence: A General Survey**, 1973
- ▶ S Russel, P Norvig: **Artificial Intelligence: A Modern Approach**, Pearson, 2022

A. M. Turing (1950) *Computing Machinery and Intelligence*, *Mind* 59: 433-466.

COMPUTING MACHINERY AND INTELLIGENCE

By A. M. Turing

I. The Imitation Game

I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think." The definitions might be framed so as to reflect so far as possible the normal use of the words, but the attitude is dangerous. If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

The new form of the problem can be described in terms of a game which we call the 'imitation game'. It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart from the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he asks either "X is A and Y is B?" or "X is B and Y is A." The interrogator is allowed to put questions to A and B thus:

C: Will X please tell me the length of his or her hair?

Now suppose X is actually A, then A must answer. It is A's object in the game to try and cause C to make the wrong identification. His answer might therefore be:

"My hair is singlet, and the longest strands are about nine inches long."

In order that tapes of voice may not help the interrogator the answers should be written, or better still, typewritten. The ideal arrangement is to have a teleprinter communicating between the two rooms. Alternatively the question and answers can be repeated by an intermediary. The object of the game for the third player (B) is to help the interrogator. The best strategy for her is probably to give truthful answers. She can add such things as "I am the woman, don't listen to him!" to her answers, but it will avail nothing as the man can make similar remarks.

We now ask the question, "What will happen when a machine takes the part of A in this game?" Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman? These questions replace our original, "Can machines think?"

Notes:

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

Motivation

- ▶ we covered **foundations of computing technology**
- ▶ we introduced basics of **algorithmic thinking and artificial intelligence learning**

open questions

- ▶ how do computers **permanently store data**
- ▶ how do **filesystems** organize data on a physical medium?
- ▶ how can we **read and write data from/to files**?
- ▶ how can we **protect data from loss**?



Mainboard

image credit: Wikipedia, User leibnizkeks, CC-BY-SA

Notes:

- **Lecture L09: Data Storage and File Systems** 07.01.2025
- **Educational objective:** We introduce technologies that allow computers to persistently store data. We discuss key functions of file systems, explore basic aspects of computer forensics and cover technologies to protect from data loss.
 - Introduction to Data Storage
 - History of Storage Technologies
 - Introduction to File Systems
 - Storage Virtualization: RAID and NAS
- **Exercise Sheet 8** 14.01.2025

Notes:

Data storage

data storage

data storage refers to the process of **recording information in a storage medium**. Storage media can have **different characteristics** in terms of volatility or persistence, durability, access times, throughput, capacity, or cost.

examples of data storage media

- ▶ Hard Disk Drive
- ▶ Floppy disk
- ▶ CD ROM
- ▶ BluRay
- ▶ SD memory card
- ▶ Solid State Disk
- ▶ DNA molecules
- ▶ carved stone



different data storage media

image credit: Wikipedia, User Zxb, CC-BY-SA

Storage size units

- ▶ in digital computers, **information is stored as binary numbers**
- ▶ **1 bit** = amount of information required to distinguish between 0 or 1 / True or False
- ▶ **1 byte** = 8 bits encoding numbers 0 – 255 ($= 2^8 - 1$)
- ▶ 1 byte = **smallest addressable unit** in (most) computer systems
- ▶ we use prefixes to denote **orders of magnitudes** of bytes of capacity/size

prefix	binary	decimal
Kilo	2^{10} (KiB)	10^3 (kB)
Mega	2^{20} (MiB)	10^6 (MB)
Giga	2^{30} (GiB)	10^9 (GB)
Tera	2^{40} (TiB)	10^{12} (TB)
Peta	2^{50} (PiB)	10^{15} (PB)
Exa	2^{60} (EiB)	10^{18} (EB)
Zetta	2^{70} (ZiB)	10^{21} (ZB)
Yotta	2^{80} (YiB)	10^{24} (YB)

IEC standard storage units

typical (permanent) storage capacities

- ▶ SD card: **dozens of GB**
- ▶ smartphone: **hundreds of GB**
- ▶ personal computer: **few TB**
- ▶ largest storage cloud: **dozens of EB**
- ▶ total storage of the world: **hundreds of ZB**

Notes:

Notes:

Time units

- ▶ **access time** = time required to access a given data object
- ▶ we use prefixes to denote **orders of magnitudes** of fractions of a second

typical time intervals in computing

- ▶ time needed to travel 1 cm at speed of light: **30 ps**
- ▶ one clock cycle of a CPU: **hundreds of ps**
- ▶ access time of cache memory: **hundreds of ps**
- ▶ access time of RAM: **tens of ns**
- ▶ access time of hard disk: **few ms**
- ▶ access time of BluRay: **tens of ms**
- ▶ time needed to travel 3.000 km at speed of light: **10 ms**
- ▶ round-trip time of packet on the Internet: **few ms** to **few hundred ms**

depending on intended application we choose different **trade-offs between characteristics of a storage medium** like, e.g., storage capacity, access time, cost, persistence, durability

prefix	factor	
pico	10^{-12} (ps)	trillionth
nano	10^{-9} (ns)	billionth
micro	10^{-6} (μ s)	millionth
milli	10^{-3} (ms)	thousandth
second	10^0 (s)	-

SI standard time units

Volatile vs. persistent data storage



Volatile Memory

- ▶ Static (SRAM) or Dynamic (DRAM, SDRAM) Random Access Mem.
- ▶ below 1 ns (SRAM) – 10 ns (SDRAM) access time
- ▶ 10 EUR per GB (SDRAM) – 500 EUR per GB (SRAM)
- ▶ bit = circuit of transistors, large and high power consumption
- ▶ used for **main/cache memory** accessed by CPU

1000 x faster than persistent memory

image credit: Wikipedia, user Evan-Amos, Public domain

Persistent Memory

- ▶ magnetic disks, SSD, tape, optical, etc.
- ▶ 100 μ s (SSD) - minutes (tape) access time
- ▶ 0.008 EUR (tape) - 0.15 per GB (SSD)
- ▶ vastly different implementations
- ▶ used for **mass storage, backup, and data archival**

1000 x cheaper than volatile memory

image credit: Wikipedia, user Evan-Amos, Public domain

Notes:

Notes:

- In the following, we will focus on different technologies for **persistent storage**

Magnetic mass storage

- ▶ **idea:** read/write patterns of magnetised blocks on a **magnetisable medium**
- ▶ in absence of external field, magnetic state of storage medium is **preserved for decades**
- ▶ since 1950s: **magnetic tape data storage**
 - ▶ first commercial use in **UNIV**ersal **A**utomatic **C**omputer (UNIVAC)
 - ▶ cheap storage for backup and archiving
 - ▶ 2025: approx EUR 0.005 per GB and up to 50 TB per cartridge
- ▶ magnetic tape provides **sequential access memory** with number-based data addressing scheme



Univac operator console and magnetic tape (ca. 1951)

image credit: Wikipedia CC-BY-SA, Rhode Island Computer Museum

Notes:

Direct Access Magnetic Storage

- ▶ sequential access memory **complicates development of applications** that interactively access data
- ▶ to simplify development of efficient software, we need **direct/random access memory**
- ▶ address specifies location of data we want to read/write
- ▶ addresses can be accessed in **any sequence**
- ▶ 1950s and 1960s: **magnetic drum memory**
 - ▶ read-write heads rotate around magnetic drum on multiple tracks
 - ▶ specific data objects can be accessed by **repositioning read-write head**
 - ▶ first commercial use in IBM 650 (17.5 kB storage)
 - ▶ in operation until 1980s



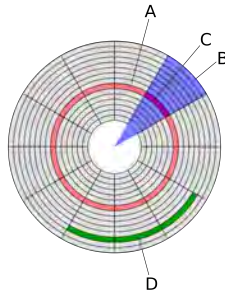
Drum memory of a Polish ZAM-41 computer

image credit: Wikipedia, public domain

Notes:

Hard Disk Drives

- ▶ since 1970s: **magnetic hard drives**
 - ▶ read/write head above rotating disk(s)
 - ▶ first commercial model IBM 350 in 1956
 - ▶ higher data density than drum drives
 - ▶ data addressable via **tracks and sectors**
 - ▶ physical acceleration can cause **head crash**
- ▶ **modern hard disk drives** (HDDs)
 - ▶ multiple disks spinning at up to 15000 rpm
 - ▶ parking of heads on power-off/during fall
 - ▶ capacity up to 32 TB
 - ▶ 512 or 4096 byte sectors common
 - ▶ approx. 3 – 10 ms access time
 - ▶ read rate ≈ 200 MB/s
 - ▶ MTBF of approx. 2 mio hours
 - ▶ approx. EUR 0.015 per GB (2025)
 - ▶ avg. life spans of approx. 2.5 mio hours



track and sector structure
of a hard disk drive

image credit: Wikipedia user Heron2/MistWiz, CC-BY-SA

Flash memory and Solid State Drives

- ▶ flash memory = transistor with electrically isolated **floating gate**
- ▶ charge in floating gate is preserved for more than 10 years
- ▶ since 1990s: **solid state drives**
 - ▶ first commercial model in 1991
 - ▶ no moving parts
 - ▶ capacity ≈ 4 TB with $100 \mu s$ access time
 - ▶ read rate ≈ 3 GB/s
 - ▶ limited write cycles ($\approx 10^4$)
 - ▶ approx. EUR 0.15 per GB
- ▶ different from HDD, data can be accessed with
(almost) the same **speed independent of location**



first commercial SSD (ca. 1991)

image credit: Public Domain, Wikipedia Commons

Notes:

Notes:

What is a filesystem?

- how can applications and users **store and retrieve data**?

challenges

- ▶ need to store **data objects** (e.g. files)
- ▶ hide **technical details of storage medium**
- ▶ data objects can **vary in size**
- ▶ data objects can **grow/shrink**
- ▶ we want to **restrict access to sensitive data**

filesystem

A **filesystem** is a software that organizes access to data based on files and folders. It hides physical details of the underlying storage medium from applications and users, stores metadata and provides access control. Filesystems are an integral part of the operating system.

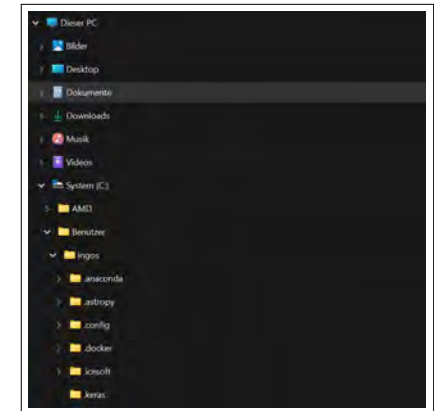
[illegible]

exemplary filesystems

- ▶ FAT, exFAT, NTFS (Windows)
- ▶ ext4, btrfs (Linux)
- ▶ APFS (Mac OS X)
- ▶ zfs (Solaris, BSD, Linux)

Volumes, files and directories

- ▶ volume = storage that is organized by **one filesystem**
- ▶ file = sequence of bytes with associated **file name and metadata**
- ▶ directory = hierarchical collection of **files and (sub-)directories**
- ▶ **Unix-style systems (Linux, Mac OS X)**
 - ▶ **single directory tree** with root node /
 - ▶ other volumes can be **mounted** into directory tree (e.g./media/cdrom)
- ▶ **Microsoft Windows**
 - ▶ volumes identified by **drive letters** (e.g. C:, D:, ...)
 - ▶ each “drive” contains separate directory tree
 - ▶ possibility to mount volumes into directory tree of another volume



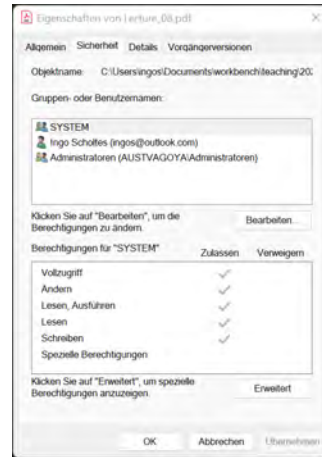
directory tree in Windows 11

Notes:

Notes:

Metadata and access control

- ▶ filesystem stores **metadata** on files and directories
 - ▶ name
 - ▶ size
 - ▶ creation time
 - ▶ last access
 - ▶ owner
 - ▶ access privileges
- ▶ we can use OS tools to set **ownership and access privileges** of files/directories
 - ▶ Windows: change via graphical interface
 - ▶ Linux: change via `chown` and `chmod` in terminal
- ▶ how does the filesystem store data and metadata associated with files and directories?



access privileges of a file

Accessing files

- ▶ operating system provides **programming interface (API)** that allows to read/write files
- ▶ we must first **“open” a file** for read and/or write access
- ▶ open function returns a **“file handle”** that can be used to access data
- ▶ we use this file handle to **read or write data** to/from file
- ▶ we finally use the file handle to **close the file**
- ▶ upon closing, operating system writes **buffered data** to storage medium (and releases file in systems with exclusive file access)

```
f = open("file.txt", "rw",  
        encoding="utf-8")  
  
text = read(f)  
write(f, "New text")  
close(f)
```

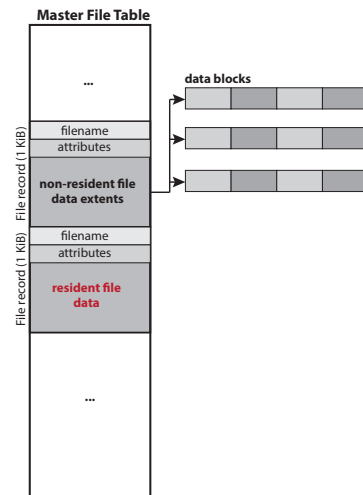
reading/writing a file (in python)

Notes:

Notes:

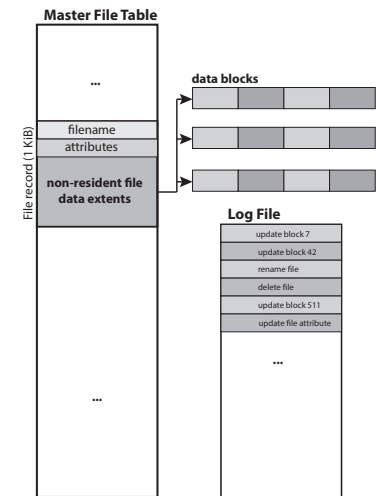
NTFS Master File Table

- ▶ **New Technology File System** (NTFS) developed by Microsoft in 1993
- ▶ standard file system in Windows operating systems
- ▶ NTFS maintains **Master File Table (MFT)** that stores MFT records (= file descriptors)
- ▶ MFT record = 1 KiB metadata on file/directory
- ▶ files can be of two different types
 1. **non-resident file** = MFT record contains addresses of (non-zero) blocks on storage medium
 2. **resident file** = MFT record contains actual data (if file is smaller than approx. 900 bytes)
- ▶ **sparse files**: blocks of zero values do not occupy disk space



NTFS Journal

- ▶ NTFS is example for a **journaling file system**
- ▶ any changes to files and metadata are recorded in a **circular log file**
- ▶ filesystem continuously processes log entries
- ▶ each log entry is processed in an **atomic** way, i.e. it is either applied completely or not at all
- ▶ in case of crash/power loss, **journal is used to recover consistent state** of file system
- ▶ like other journaling filesystems (e.g. APFS, ext4) NTFS is **self-healing**



Notes:

Notes:

Practice session

- ▶ we create small files on an NTFS volume
- ▶ we explore the behaviour of NTFS regarding **resident and non-resident files**

```
Contents of section 'text':
0530 31e48889 d1e4889 d2883a4 090554c 3.T..M..H...PTL
0540 8a058a01 0000488d 0d130100 00488d3d .....H..*
0550 e0000000 ff1580a0 2000f40f 1f400000 .....D..
0560 488d3d09 0a200055 48809a1 0a200048 H...LM...H
0570 39f84889 a5741948 8a055a0a 20004885 9.H..t.H..Z..H
0580 c0740d5d ffa0662a 0f2f8400 00000000 ..}.f.....
0590 5dc30f1f 4000662a 0f2f8400 00000000 ].-B.f.....
05a0 488d3d09 0a200048 8d35620a 20005548 H..L..H.Sb..LM
05b0 29fe4889 a548c1fe 03a889f0 48c1a83f }.H..M..H..P
05c0 4801c048 d1f47418 4800921 0a200048 H..M..L.H..L..H
05d0 85c0740c 5dff4066 0f2f8400 00000000 ..t}.f.....
05e0 5dc30f1f 4000662a 0f2f8400 00000000 ].-B.f.....
05f0 083d3d0a 20000075 2f488d3d f70f2000 +..c..u.H..*
0600 00554889 a5740c48 0d3d9a09 2000a88d LM..t.H..*
0610 ffffffe8 48ffffff c085f109 2000015d .....H.....}
0620 c30ff4f0 00000000 f23600f 1f400000 .....f..D..
0630 554809a5 5de96aff ffff5548 00a54883 LM..}f...LM..H
0640 ec10488d 05900000 004889a5 f8480d45 ..H.....M.E.H.E
0650 f04889c7 a8d7efff ff000000 0000c2c3 H.....
0660 4157415d 4900f7a3 55a1544c 0d3d4a0f 0000f..A00T..SE
0670 20005548 8d2d4607 20005541 89f04089 LM..F..SA..T
0680 f64c29a5 4803a000 48c1f003 a857feff ..L.H..H...M..
0690 f48d85ed 742015db 0f2f8400 00000000 ..t..t.....
06a0 4c89f9e0 89f64a09 a741ff14 d04883c3 L..L..D..A...H..
06b0 0148396d 75aa4883 c4883c5d 415c415d ..B..u.H...[A]A
06c0 415a415f c300662a 0f2f8400 00000000 A'A..f.....
06d0 f3c3
```

practice session

see directory 09-02 in gitlab repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_WiSe_CS_Practice

NTFS MFT slack and computer forensics

Consider a user storing a **100 byte text file that contains a list of passwords**. He/she then deletes the file and stores a 20 MB image file immediately afterwards. Discuss **whether the deleted passwords can be recovered** in the context of an NTFS filesystem.

If a **resident file** is deleted, the corresponding file record in the MFT will go out of use. A reuse of the record for a non-resident file can lead to **MFT slack**, i.e. not all of the record's 1024 bytes are overwritten by metadata of the new file. By inspecting the MFT table (which can be accessed like a regular file on the disk) we can possibly recover the data.

Notes:

Notes:

Preventing data loss

reasons for data loss

- ▶ human error, e.g. accidental deletion, liquid spilling, etc.
- ▶ theft, sabotage, wilful deletion
- ▶ failure of storage medium or controller
- ▶ data corruption due to power failure, crash, etc.
- ▶ computer virus or ransomware
- ▶ fire or natural disasters

- ▶ how can we **protect data against loss**?

- ▶ protection against data loss requires **redundant storage of data**

- ▶ redundancy = increased cost of HW, electricity, maintenance, etc.



image credit: www.infosecurity-magazine.com

Data backups

data backup

A data backup is a **copy of computer data taken and stored elsewhere** so that they **may be used to restore the original** after a data loss event. *adapted from Wikipedia*

- ▶ backup (ideally) protects from **all possible causes for data loss**
- ▶ assume that **loss of original data** occurs with prob. p and **loss of backup copy** occurs with prob. q
- ▶ if loss events are **independent**, total loss probability with n backups is

$$P(\text{data loss}) = p \cdot q^n$$

- ▶ can we achieve **full independence**?



image credit: [Wikimedia Commons, Renee Comet, public domain](https://commons.wikimedia.org/wiki/File:Reene_Comet.jpg)

Notes:

Notes:

Backup types

- ▶ we often store backups on **different** (potentially cheaper) storage medium, e.g. tape/optical backup of disk or disk backup of SSD
- ▶ to protect from theft, fire and natural hazards, backups should be **stored in an off-site location**
- ▶ **full backup**: full copy of original data (potentially including OS image)
- ▶ **incremental backup**: periodically copy data that have changed since last (incremental) backup

online backup

backup continuously attached to original source

- ▶ example: NAS continuously available on network
- ▶ facilitates incremental backups at small intervals
- ▶ fast restoring in case of error
- ▶ but: prone to accidental/purposeful deletion

offline backup

backup not continuously attached to original source

- ▶ example: tapes/disks stored in a shelf
- ▶ data not accessible except during backup
- ▶ safe from accidental/purposeful deletion
- ▶ but: restoring of data requires additional action

Notes:

Storage virtualization

- ▶ **filesystem abstracts from physical location of data** on a storage medium
- ▶ we can extend this to multiple storage media, i.e. provide *logical* volume that can span multiple devices
- ▶ actual location/organization of data is transparent to the user
- ▶ provision of a single logical view of storage distributed across disks or computers is called **storage virtualization**

examples

- ▶ logical volume accessed on a server machine via network (network-attached storage, NAS)
- ▶ logical volume spanning multiple disks on a single computer (RAID)



Synology NAS DS1821+

image credit: www.synology.com

Notes:

RAID storage

- ▶ RAID = redundant array of inexpensive (or independent) disks
- ▶ increase performance and/or reliability by storing data to multiple disks (HDD or SSD)
- ▶ RAID can be implemented in hardware or software
- ▶ depending on **RAID levels** we can tolerate loss of one or more physical disks

RAID levels

- ▶ RAID 0 (striping)
- ▶ RAID 1 (mirroring)
- ▶ RAID 1+0 (mirroring + striping)
- ▶ RAID 5 (striping with distributed parity block)
- ▶ RAID 6 (striping with two distributed parity blocks)



Synology NAS DS1821+

image credit: www.synology.com

RAID levels 0 and 1

RAID 0

- ▶ **striping of data** across two or more physical disks
- ▶ requires $n \geq 2$ disks
- ▶ storage space efficiency factor = 1, i.e. sum of all disk sizes
- ▶ increases speed of read/write operations
- ▶ no redundancy, i.e. **no protection against disk failures**

RAID 1

- ▶ **mirroring of data** across two or more physical drives
- ▶ requires $n \geq 2$ disks
- ▶ storage space efficiency factor = $\frac{1}{n}$
- ▶ increases speed of read/write operations
- ▶ protects against $n - 1$ **disk failures**

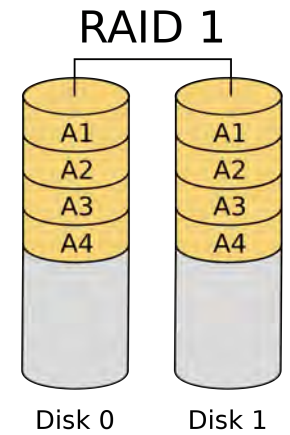


image credit: Wikipedia, User Cburnett, CC-BY-SA

Notes:

Notes:

RAID levels 5 and 6

RAID 5

- ▶ **striping of data** with distributed parity block
- ▶ parity bits store whether sum of bits in other stripes is **odd or even**
- ▶ requires $n \geq 3$ disks
- ▶ increases speed of read/write operations
- ▶ storage space efficiency factor = $1 - \frac{1}{n}$
- ▶ **protection against single disk failure**
- ▶ but: volume rebuild after replacement of faulty drive can take a long time

RAID 6

- ▶ **striping of data** with two distributed parity blocks
- ▶ requires $n \geq 4$ disks
- ▶ increases speed of read/write operations
- ▶ storage space efficiency factor = $1 - \frac{2}{n}$
- ▶ **protection against simultaneous failure of two disks**
- ▶ with RAID 6 we can tolerate failure of one more disk while the volume is rebuilt

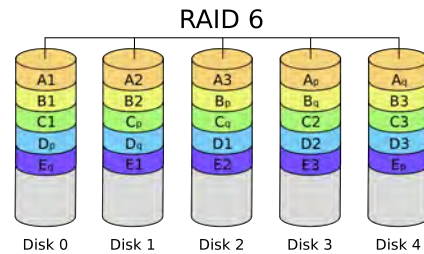


image credit: Wikipedia, User Cburnett, CC-BY-SA

A practical example



QNAP TS-869 Pro Network Attached Storage (NAS)

8 × 10 TB HDDs, one disk used as **hot spare**

7 HDDs used in a single **RAID 6 volume**
usable volume size = $(1 - \frac{2}{7}) \cdot 70 \text{ TB} \approx 42 \text{ TB}$

protected against simultaneous failure of two HDDs

can be accessed like local drive via SMB, NFS or AFP protocol on Windows, Linux, or Mac

image credit: www.qnap.com

Notes:

Notes:

In summary

- ▶ we have covered different **storage technologies** used in computing
- ▶ storage media have **largely different characteristics** that qualify them for different applications
- ▶ we explored how **file systems** organize physical storage of data and how they provide **file and directory abstractions**
- ▶ we discussed backup and storage virtualization techniques that help to **protect from data loss**



Paintings in the Chauvet cave (approx. 30.000 BC)

[image credit: Wikipedia, User HTC, Public domain](#)

Self-study questions

1. What are the key advantages and disadvantages of persistent storage technologies like tape, HDD, or SSDs?
2. What is the difference between sequential, direct and random access memory. Give one example for storage technologies for each access mode.
3. How can we address data on a hard disk drive?
4. What are the key functions of a filesystem?
5. Give three examples for filesystems in modern operating systems.
6. Explain the terms volume, directory, and file in the context of filesystems.
7. Explain the function of the Master File Table in the NTFS filesystem?
8. Why does a journaling filesystem provide better data consistency in case of a sudden power loss?
9. Which of the RAID levels provide tolerance against at least one failing drive?
10. What is the advantage of RAID 6 over RAID 1?
11. In which specific scenario does RAID 6 have advantages over RAID 5?
12. Can the use of a RAID 6 volume replace a backup? Explain your answer.

Notes:

Notes:

Literature

reading list

- ▶ **Timeline of Computer History: Memory and Storage**
<https://www.computerhistory.org/timeline/memory-storage/>
- ▶ David Both: **An Intorudction to Linux File Systems**, Medium, 2016
<https://opensource.com/life/16/10/introduction-linux-file-systems>
- ▶ CK Wee: **Analysis of hidden data in the NTFS file system**, 2006
<https://www.forensicfocus.com/articles/analysis-of-hidden-data-in-the-ntfs-file-system/>
- ▶ **Internals and Basics of the NTFS File System** (with discussions of general HDD and filesystem basics)
<https://www.ntfs.com>
- ▶ **Wikipedia article: Backup**
<https://en.wikipedia.org/wiki/Backup>
- ▶ **Wikipedia article: RAID**
<https://en.wikipedia.org/wiki/RAID>

Notes:

- Following your feedback in the lecture evaluation, this week's reading list focuses on freely available online sources.

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

Motivation

- ▶ filesystems allow applications to **store data in files**
- ▶ but: structure/meaning of these files is determined by application/user
- ▶ how can we **store information** in a structured, interoperable, and consistent way?
 - ▶ customer accounts in a bank
 - ▶ hotel reservations
 - ▶ posts/media in social media platforms

we use **database systems** to organize and store information and to facilitate fast knowledge extraction by multiple users and applications



archive with boxes

image credit: Archivio-FSP, CC BY-SA 3.0

- ▶ efficient database systems are **key technology** in modern information technology

Notes:

- **Lecture L10: Database Systems** 14.01.2025
- **Educational objective:** We introduce basic concepts of database systems. We use an example to study relational database design, introduce the query language SQL and motivate database transactions.
 - Introduction to Relational Databases
 - Relational Database Design
 - Structured Query Language (SQL)
 - Database Transactions
- **Exercise sheet 9** 21.01.2025

Notes:

Database Management Systems

- ▶ **database (DB)**: collection of data on a given “mini-world”, a part of the world that is of interest for our database

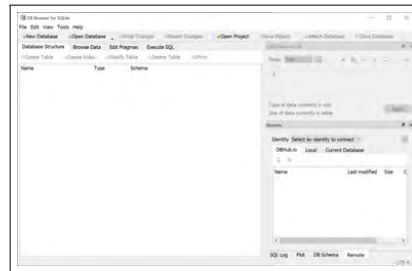
example mini-world

We can consider the **mini-world of movies**, i.e. we want to store information on a movie collection consisting of movies, actors, and directors.

- ▶ **database management system (DBMS)**: software that allows to create, manage, and query a DB

key functionality of a DBMS

- ▶ allow user to **define databases (DB)**
- ▶ allow users to **insert, modify and delete data** into a DB
- ▶ support efficient **retrieval of information** from the DB
- ▶ guarantee **consistency of data**
- ▶ manage **concurrent access** by multiple users
- ▶ **user authentication** and access auditing
- ▶ support **transactions**



Management interface DB Browser for DBMS SQLite

Relational Databases

- ▶ there are different **types of database management systems**

- ▶ hierarchical or graph-based DBMS
- ▶ object-oriented DBMS
- ▶ relational DBMS

- ▶ relational database consists of **relations (= tables)** that consist of **tuples (= table rows)**

- ▶ tables have **attributes (= columns)** that must adhere to a given **data type**, e.g. variable-length text, integer or floating point number, date/time, etc.

- ▶ how could we store information on a given mini-world in multiple tables?

example table

Movie		
Title	Year	Genre
Clockwork Orange	1971	Dystopia
Samsara	2011	Documentary
Woman of Straw	1964	Crime
Highlander	1986	Phantasy

example for a table **Movie** with three attributes **Title, Year, Genre** and four rows, representing four movies

Notes:

- We first briefly introduce some key terms and concepts.
- With the term database we refer to a specific collection of data of interest that we want to store on a so-called “mini-world”. Such a mini-world is one (tiny) part of the world that we want to model with our database, e.g. the customers of a bank, all users and posts of a social media platform, or the movies stored in a movie collection.
- While a database is one specific collection of data, we call the software that allows us to create, manage and query such databases a database management systems (DBMS). A single DBMS can manage many different (maybe thousands of) databases.
- A DBMS is an example of a so-called middleware, i.e. a software that sits between the operating system and other applications that use functions provided by the middleware. As such, the functionality provided by a DBMS goes way beyond what is provided by the filesystem or the OS. It allows us to define databases, insert, modify and delete data, it supports the efficient retrieval of data and guarantees the consistency of data. It also manages concurrent access, allows certain users to only access some parts of a database and supports transactions (later more on this).
- While we could build some functions based on files, DBMS provide much more convenient functions to efficiently retrieve data from large databases.

Notes:

- There are different types of DBMS that are based on different concepts how the data are organized or stored. For instance we could store data as a graph or network, where nodes store values and links connected those values. Or we could store data in a hierarchical tree structure, where each record is again linked to other records. Or we could store a collection of objects that have clearly defined types.
- In this lecture, we will consider relational databases, which are arguably among the most important DBMS. The key idea is that data is stored in a table format, where each table has multiple columns, and each row represents one tuple (or record). We call such a table also a relation, because each row relates the values of different columns to each other.

Database schema vs. instance

- ▶ **relational database schema** describes the “data model” of a database
 - ▶ tables
 - ▶ attributes of each table
 - ▶ data type of each attributes
- ▶ **database schema** is usually **created once** when we define a database
- ▶ **database instance** refers to data stored in a database at a given point in time
 - ▶ collection of all data stored in tables
 - ▶ database instance changes whenever contents is updated

simple relational database schema with four attributes

Movie			
Title (VARCHAR)	Year (INT)	Genre (VARCHAR)	Length (INT)

instance 1

Movie			
Title (VARCHAR)	Year (INT)	Genre (VARCHAR)	Length (INT)
Woman of Straw	1964	Crime	116
Highlander	1986	Phantasy	111

instance 2

Movie			
Title (VARCHAR)	Year (INT)	Genre (VARCHAR)	Length (INT)
Clockwork Orange	1971	Dystopia	131
Woman of Straw	1964	Crime	116

A (badly designed) relational database

consider a database for a **mini-world of movies**, where you want to store information on the title, year, length and genre of movies, as well as on directors and actors.

relational database

Movie					
Title	Year	Genre	Length	Director	Actor
Clockwork Orange	1971	Dystopia	131	Stanley Kubrick	Malcolm McDowell
Samsara	2011	Documentary	102	Ron Fricke	NULL
Woman of Straw	1964	Crime	116	Basil Dearden	Gina Lollobrigida
Woman of Straw	1964	Crime	116	Basil Dearden	Sean Connery
Highlander	1986	Phantasy	111	Russell Mulcahy	Sean Connery
Highlander	1986	Phantasy	111	Russell Mulcahy	Christopher Lambert

question

do you see any problems with this relational database schema consisting of a single table?

Notes:

- We can think of different structures of such table in terms of which columns exist and what are the types of these columns. This is the so-called schema, which originates from Greek $\sigma\chi\eta\mu\alpha$ “schema” translating to “shape”.
- The schema of a database is typically created once when the database is created and it is changed very rarely (if at all). For a good schema, it should not be necessary to change it as new data is stored.
- A database instance refers to the actual tables stored in a database, which are consistent with a schema. So when we update data in a database, the database instance changes but the schema remains the same.

Notes:

- How can we design a database schema for a specific purpose? Let us consider an example of a DB for a mini-world of movies, where we want to store information on movies, actors and directors.
- A simple solution would be to define a single table that holds all of these information.
- Is this a meaningful approach?

Good relational database design?

- ▶ during **database design** we determine tables and their schemas
- ▶ good database design ensures that attributes are **atomic**, such that we can easily query data (e.g. first- or lastname)
- ▶ what if we want to add multiple actors to a single movie?
 - ▶ repeated Actor attributes?
 - ▶ multiple rows with different actors?
- ▶ good database design tries to **eliminate redundancy**, i.e. information that is stored multiple times
 - ▶ uncontrolled redundancy leads to **data inconsistency**
 - ▶ any unavoidable redundancy must be **managed and controlled by DBMS** to avoid inconsistencies
- ▶ how can we design a **good database schema**?

atomic attributes

attribute **Director** is non-atomic as it consists of (divisible) First- and Lastname

Movie	
Title	Director
Clockwork Orange	Stanley Kubrick

repeated attributes

Actor1 and Actor2 are repeated

Movie		
Title	Actor1	Actor2

redundancy

Title and Year are redundant

Title	Year	Actor
Highlander	1986	Connery
Highlander	1986	Lambert

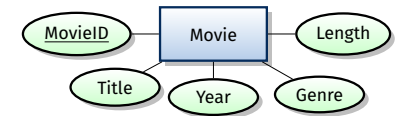
ER model: Entities

- ▶ we can use **entity-relationship models** to systematically design database schema
- ▶ we use **ER-diagrams** to visualize ER models
- ▶ we use **entities** to model specific “things of interest” in our mini-world

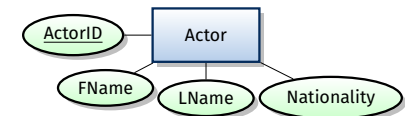
example: Entities in movie mini-world

- ▶ **MOVIEs**
- ▶ **ACTORs**
- ▶ **DIRECTORs**

- ▶ each entity can have multiple **attributes**
- ▶ how can we **uniquely identify** a given entity?
- ▶ **key attributes** must have unique values across all entities of a given type



entity **Movie** with four attributes
(+ key attribute **ID**)



entity **Actor** with three attributes
(+ key attribute **ID**)

Notes:

- A key challenge in database design is to determine a meaningful schema that avoids inconsistencies and redundancies and makes it easy to query the data. Let us consider this in our example.
- A first issue is that we have non-atomic attributes, i.e. we have attributes like Director that could be further divided into a first and a last name. Storing data in non-atomic form makes it difficult to query data, e.g. if you want to search for the lastname.
- A second issue is that we may want to add multiple actors to a single movie. There are two ways to do this if we use a single table.
- We could first repeat the attribute actor multiple times, which is a bad idea because this means that we either have to change the schema when we add an actor or we have to add as many columns as we can have actors (thus defining a maximum), where we have a lot of empty cells for movies that do not have that many actors.
- A second solution would be to repeat a row for each actor, which would introduce redundancy. Redundancy is a very bad idea for databases, not only because it is inefficient to store the same data multiple times, but even more important because we could end up changing one value while keeping its copy unchanged. This would lead to inconsistent data, which we want to avoid at all cost!
- Good relational database design thus avoids redundancy. Those redundancies that we cannot avoid should be declared to the DBMS, so the DBMS can control that no inconsistencies can ever occur.
- But how can we systematically create a good schema for a database that avoids those issues?

Notes:

- The Entity-Relationship (ER) model helps us to systematically design good database schema (especially relational schema). An ER model consists of two different components: entities and relationships. We visualize an ER model via so-called ER diagrams, where specific symbols represent entities and relationships.
- **Entities** model the types of “things” that we store information on. In our example, these could be movies, actors, and directors. We typically use the singular form as entity name. In an ER diagram, entities are typically drawn as rectangular boxes.
- Each entity can have multiple associated **attributes** that store the actual values for a given entity. Attributes are drawn as ovals connected to the associated entity by a line.
- Some of the attributes of an entity can be **key attributes**, which are assumed to be take unique values among all entities of a given type. As an example, there are multiple different movies that have the same name (e.g. “The Fugitive” which was released in 1947 and in 1993), so we cannot use the name of a movie to unambiguously refer to a specific movie. We could instead assign a unique number (i.e. an identifier or ID) that we define as key. In the example, we could now assign the 1947 version of “The Fugitive” a unique ID 42, while the 1993 version gets a different ID 125.

ER model: Relationships

- in an ER model, we additionally model **relationships** between entities

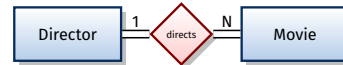
example: relationships in movie mini-world

- DIRECTOR **directs** MOVIE
- ACTOR **plays in** MOVIE

- cardinality constraint** limit with how many other entities an entity can at most be related
- we can use **participation constraint** to require entity to “participate” in relationship with other entity

example: cardinality and participation constraints

- each movie is directed by **exactly one** director
- each director must direct **at least one** movie
- each movie can have **any number** of actors
- each actor must play in **at least one** movie



One-to-Many relationship **directs**
between entities **Director** and **Movie**



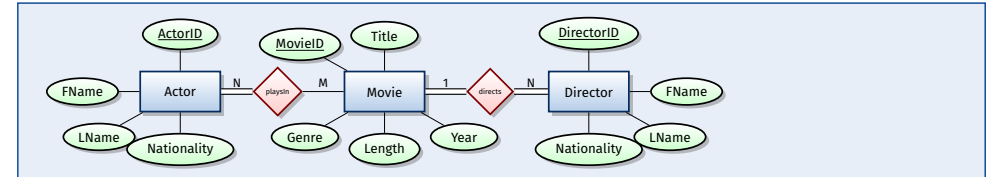
Many-to-Many relationship **plays in**
between entities **Actor** and **Movie**

constraints in ER diagrams

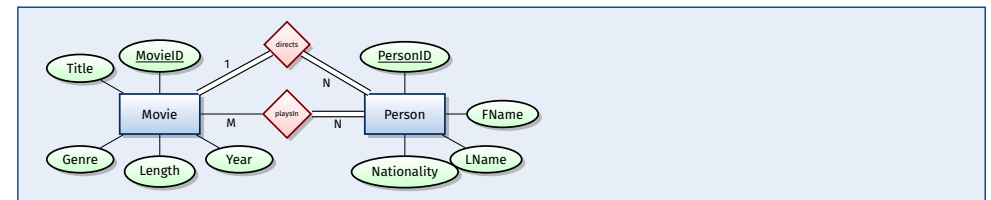
in ER diagrams, cardinality constraints are indicated by numbers (1, *N*, *M*), while participation constraints are indicated by a double line. We read cardinality constraints from entity via relation to number (e.g. each “Movie” is “directed by” at most “one” director)

Group exercise

- Create an Entity-Relationship diagram for a **mini-world of movies**, where you want to store information on the title, year, length and genre of movies, as well as on the names and nationalities of directors and actors.



- What happens if an actor also directs a movie? Do we need two entities Actor and Director.



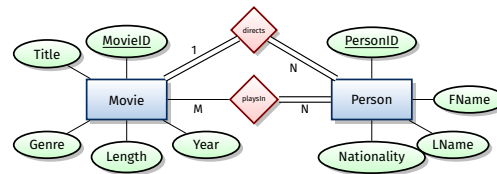
Notes:

- Apart from entities, in the ER model we also model **relationships** between entities. In an ER diagram relationships are typically visualized by a diamond shape, which is connected to the associated entities. In the example above, we see two relationships “directs” and “playsIn” that connect the entities Director and Movie as well as Actor and Movie, respectively.
- A relationship between entities of type A and B can be subject to additional constraints that restrict with how many entities of type B each entity of type A is minimally or maximally related (and vice-versa). Cardinality constraints restrict the maximal number, i.e. we can add a restriction that allows each movie to only have a single director, or that allows one movie to have any number of actors.
- In an ER diagram, we indicate **cardinality constraints** by the small numbers next to a relationship. We read this in the director “Entity, Relationship, Cardinality Constraint”, i.e. in the example above each director can direct *N* (i.e. any number of) movies. In the reverse direction, we read that each movie can be directed by at most director.
- In addition to cardinality constraints, we can also set **participation constraints**, i.e. we specify that each entity must participate in the relationship, i.e. that there must at least be one relation to an entity of the other type. In an ER diagram we indicate this by a **double line** connecting the participating entity and the relationship, i.e. in the example above, each director must direct at least one movie and each movie must have at least one director (together with the cardinality constraint that means each movie must have **exactly one** director). In the example below, each actor must play in at least one movie (and he/she can play in any number of movies), while movies can have any number of actors (including no actors at all, since there is no double line from the entity Movie to the relationship playsIn).

Notes:

Translating ER models to relational schema

- ▶ to avoid redundancy, we store **information on different entities in different tables**
 - ▶ example: Movie, Person
- ▶ to avoid non-atomic attributes, we **store divisible information in multiple attributes**
- ▶ **primary key** (underlined attribute) uniquely identifies each row in the table (e.g. via an auto-incremented number)
- ▶ we use additional tables to store **relationships between entities**
 - ▶ example: directs, playsIn
 - ▶ **foreign keys** refer to primary keys in tables Movie and Person
 - ▶ combination of (PersonID, MovieID) is primary key in tables directs and playsIn



relational schema implementing ER model

Movie				
<u>MovieID</u>	Title	Year	Genre	Length

Person			
<u>PersonID</u>	FirstN	LastN	Nationality

directs	
<u>PersonID</u>	<u>MovieID</u>

playsIn	
<u>PersonID</u>	<u>MovieID</u>

Example for corresponding database instance

database instance

Movie				
<u>MovieID</u>	Title	Year	Genre	Length
1	Clockwork Orange	1971	Dystopia	131
2	Samsara	2011	Documentary	102
3	Woman of Straw	1964	Crime	116
4	Highlander	1986	Phantasy	111

Person			
<u>PersonID</u>	FirstN	LastN	Nationality
1	Malcolm	McDowell	England
2	Sean	Connery	Scotland
3	Christopher	Lambert	USA
4	Stanley	Kubrick	England
5	Ron	Fricke	USA
6	Russel	Mulcahy	Australia
7	Basil	Dearden	England

playsIn	
<u>PersonID</u>	<u>MovieID</u>
1	1
2	3
2	4
3	4

directs	
<u>PersonID</u>	<u>MovieID</u>
4	1
5	2
6	3
7	4

Notes:

- Once we have an ER model of our database, we must translate it to a relational schema. Luckily, this works in a more or less automatic fashion and it does not require any creativity!
- We can simply create one table for each entity and relationship type in our ER model. The attributes of the entities are the columns of our tables. We also use the key attributes as primary keys, which uniquely identify entities.
- For the relationships, we create tables that use foreign keys to refer to the primary keys of entities that participate in the relationship, i.e. the information that a given actor plays in a movie would just be stored by a row that relates the PersonID of the actor to the MovieID of the movie. We could now either introduce a new ID for each of those relations, or we can just use the combination of foreign keys as primary key (since each relation can occur only once).
- Note that for a N:M relation we can just store multiple rows, where each row can relate the same Actor (i.e. a given PersonID) to multiple Movies (i.e. different MovieIDs) and vice-versa.

Notes:

Structured Query Language (SQL)

- ▶ how can we define, manipulate, and query our database in practice?
- ▶ we use **special “programming languages”** to interact with databases
 - ▶ data definition language (DDL) to create database schema, define data types, set keys, etc.
 - ▶ data manipulation language (DML) to insert, update, and query data
- ▶ SQL is **standard language of many DBMS**
 - ▶ SQL = Structured Query Language
 - ▶ pronounced: “SEQUEL”
 - ▶ comprises both DDL and DML
- ▶ SQL is a **declarative language**, i.e. we declare **which** data we want rather than **how** DBMS should perform the query

example database query (SQL)

```
SELECT * FROM movie
WHERE YEAR = 1986;
```

SQL-based relational DBMS

- ▶ Oracle DBMS
- ▶ SAP HANA
- ▶ PostgreSQL
- ▶ MySQL
- ▶ SQLite

SQL DDL: Creating tables

Defining tables

- ▶ we can use the **CREATE TABLE** statement to define schema of tables
- ▶ **PRIMARY KEY** ensures that attribute value(s) uniquely identify a row
- ▶ **FOREIGN KEY** ensures that referenced rows exist in other table

create table Movie with primary key

```
CREATE TABLE Movie (
  MovieID INTEGER,
  Title CHAR(30),
  YEAR INTEGER,
  Genre CHAR(30),
  Length INTEGER,
  PRIMARY KEY (MovieID)
);
```

creates new table *Movie* with following schema

Movie				
MovieID	Title	Year	Genre	Length

create table directs with foreign keys

```
CREATE TABLE directs (
  PersonID INTEGER,
  MovieID INTEGER,
  FOREIGN KEY (PersonID)
    REFERENCES Person(PersonID),
  FOREIGN KEY (MovieID)
    REFERENCES Movie(MovieID),
  PRIMARY KEY (PersonID, MovieID)
);
```

creates new table *directs* with following schema

directs	
PersonID	MovieID

Notes:

Notes:

SQL DML: Insert, update and delete data

Manipulating data

- ▶ we can use **INSERT INTO**, **UPDATE** and **DELETE FROM** to manipulate rows
- ▶ **WHERE** can be used to specify which rows should be updated/deleted

example

- ▶ add row to table movie

```
INSERT INTO Movie(MovieID,
YEAR, Title, Genre,
Length)
VALUES (1, 2001,
"The Lord of the Rings",
"Phantasy", 228);
```

- ▶ or equivalently

```
INSERT INTO Movie
VALUES (1,
"The Lord of the Rings",
2001, "Phantasy", 228);
```

example

- ▶ change title of movie

```
UPDATE Movie
SET Title =
"The Fellowship of the Ring"
WHERE MovieID = 1;
```

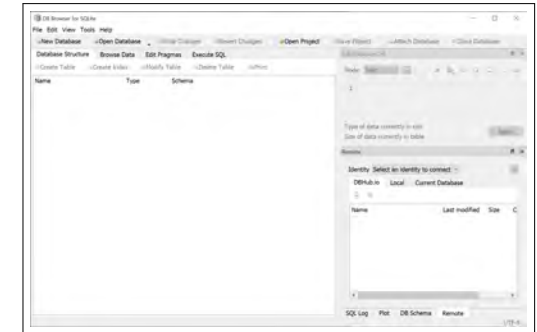
example

- ▶ delete all movies released in 2001

```
DELETE FROM Movie
WHERE YEAR = 2001;
```

Practice Session

- ▶ we introduce the lightweight SQL-based relational DBMS **SQLite**
- ▶ we use the **DB Browser for SQLite** to define the movie database using SQL DDL statements
- ▶ we use SQL DML statements to **insert and modify data into our movie DB**



practice session

see directory 10-01 in **gitlab** repository at

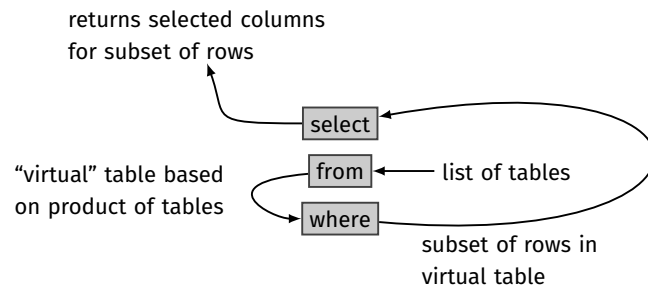
→ https://gitlab2.informatik.uni-wuerzburg.de/ml4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

Using SQL to query data

- ▶ simple SQL query consists of **SELECT** statement with **FROM** clause and optional **WHERE** clause
- ▶ for this lecture we ignore optional **GROUP** and **HAVING** clauses that can be used to calculate aggregates over grouped rows
- ▶ simple SQL queries with **SELECT**, **FROM** and **WHERE** are processed as follows:



SQL: Evaluation of a simple query

1. compute **product of tables** listed in **FROM** clause

X						
	A	B	C			

Y						
	D	E				

Z						
	F					

FROM X, Y, Z

A	B	C	D	E	F

2. eliminate rows that do not satisfy **WHERE** clause

A	B	C	D	E	F
					1
					1
					1
					17
					...
					99
					5

WHERE F = 1

A	B	C	D	E	F
...	1
...	1
...	1
...	1

3. return rows for columns listed in **SELECT** clause

A	B	C	D	E	F
...	1
...	1
...	1

SELECT A, C, E, F

A	C	E	F
...	1
...	1
...	1

Notes:

Notes:

SQL Query Examples

Querying data

- ▶ we use **SELECT** statement to **query a database**
- ▶ column list after **SELECT** can be used to reduce result certain columns
- ▶ **FROM** can be used to define (virtual) table from which data is queried, possibly combining rows from multiple tables
- ▶ **WHERE** can be used to specify condition that selected rows must satisfy

example: query with WHERE clause

- ▶ return Title and Year of all Phantasy movies before 2001

```
SELECT Title, YEAR FROM Movie
WHERE Genre = "Phantasy"
AND YEAR < 2001;
```

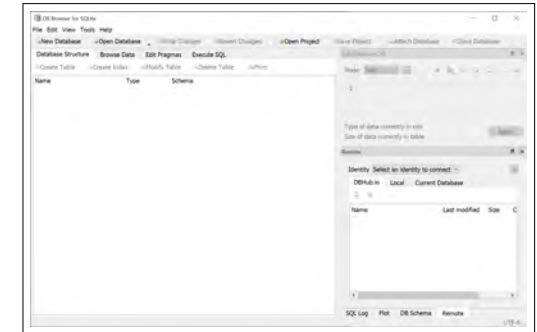
example: query that joins multiple tables

- ▶ join Movie, directs, Person tables and select last name of all directors of Phantasy movies

```
SELECT Person.LName FROM
Movie, directs, Person
WHERE Genre = "Phantasy"
AND Movie.MovieID =
directs.MovieID
AND Person.PersonID =
directs.PersonID;
```

Practice Session

- ▶ we use SQL queries to retrieve data from our database
- ▶ we show how we can use the **SELECT** statement to join data from multiple tables
- ▶ we demonstrate how the **JOIN** keyword simplifies the joining of data from multiple tables



practice session

see directory 10-02 in gitlab repository at

→ https://gitlab2.informatik.uni-wuerzburg.de/m4nets_notebooks/2024_wise_infhaf_notebooks

Notes:

Notes:

From database queries to transactions

- ▶ we finally consider a relational database that holds data on **customer accounts in a bank**
- ▶ wire transfer of EUR 5000 between two accounts can be implemented as **sequence of two SQL queries**
- ▶ **what could possibly go wrong?**
 1. concurrent query of database (after first, but before second **UPDATE** operation)
 2. DBMS could crash after first UPDATE statement

example queries

```
UPDATE account SET
  balance = balance - 5000
WHERE IBAN = "DE70 32 4134 1232 1231";
UPDATE account SET
  balance = balance + 5000
WHERE IBAN = "DE70 32 3521 4211 1124";
```

transactions

We call a sequence of database operations (e.g. insert, deletion, modification, retrieval) that form a **logical unit a transaction**.

ACID property and SQL Transactions

- ▶ for transactions we must guarantee **ACID property**
 - A **Atomic** = indivisible, i.e. either all operations succeed or none
 - C **Consistent** = integrity of data is not violated
 - I **Isolated** = concurrently executed transactions do not have side-effects
 - D **Durable** = result of successful transaction is stored permanently, i.e. it will not be reverted
- ▶ in SQL we use **BEGIN** and **COMMIT** to **group queries belonging to transaction**
- ▶ any concurrent query executed before **COMMIT** see the DB state *before* the transaction was started
- ▶ if any operation within transaction fails, all changes within that transaction will be rolled back

example transaction

```
BEGIN;
UPDATE account SET
  balance = balance - 5000
WHERE accNr = "DE70 32 4134 1232 1231";
UPDATE account SET
  balance = balance + 5000
WHERE accNr = "DE70 32 3521 4211 1124";
COMMIT;
```

Notes:

Notes:

In summary ...

- ▶ we motivated **database management systems**, which are implemented on top of file systems
- ▶ we considered key concepts of **relational database design**
- ▶ we introduced the **query language SQL** and showed how we can use it to query information in a relational database
- ▶ we motivated the need for **database transactions** and introduced ACID properties



drum memory of ZAM-41 computer (ca. 1961)

[image credit](#): Public Domain, Wikipedia Commons

Self-study questions

1. Explain the components of a relational database.
2. Give an example for a database schema that leads to redundancy and potentially inconsistencies.
3. Give an example for a table schema with a non-atomic attribute.
4. How can we avoid redundancy in a relational database schema?
5. Given a simple ER model for the mini-world of a company, storing information on employees and projects.
6. Explain the difference between a primary key and a foreign key?
7. Explain how we can represent an N:M relationship between entities in a relational table.
8. Give an example for an ER model with a One-to-One (1:1) relationship.
9. Explain the difference between cardinality and participation constraints.
10. How can we merge information on related entities using SQL queries?
11. Explain the challenges that can occur if multiple database queries belong to a transaction.
12. What are the ACID properties?
13. How can we define database transactions in SQL.

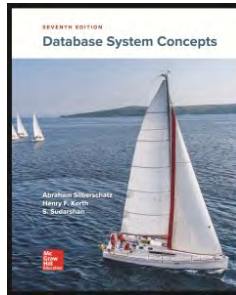
Notes:

Notes:

References and credits

reading list

- ▶ Abraham Silberschatz, Henry F. Korth, S. Sudarshan: **Database System Concepts**, 7th Edition, McGraw Hill Education with online material
→ <https://www.db-book.com/>
- ▶ SQL Tutorial
→ https://sqlzoo.net/wiki/SQL_Tutorial
- ▶ W3Schools SQL tutorial
→ https://www.w3schools.com/sql/sql_intro.asp
- ▶ Online SQL Interpreter
→ <https://www.db-book.com/university-lab-dir/sqljs.html>



slides reuse material kindly provided by

- ▶ **Prof. Dr. Michael Böhlen**
University of Zürich, Switzerland
- ▶ **Prof. Dr. Johann Gamper**
Free University of Bozen-Bolzano, Italy

Notes:

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

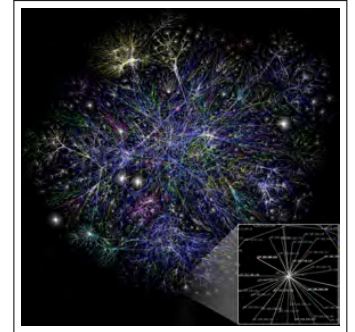
ingo.scholtes@uni-wuerzburg.de

Motivation

- ▶ we studied modern **computing hardware** and introduced key concepts **to store, manage, and process information**
- ▶ success of digital technologies is due to **ability to communicate**
 - ▶ **human-human** communication
→ voice, video, chat, E-Mail, ...
 - ▶ **human-machine** communication
→ search engine, AI chatbots, smart home, ...
 - ▶ **machine-machine** communication
→ smart grid, smart manufacturing, IoT, ...

open questions

- ▶ how does **telecommunication** actually work?
- ▶ how can we use **communication networks** to exchange information between a large number of devices?
- ▶ how does **global Internet communication** work?



partial map of the Internet as of January 15, 2005

image credit: Opte project, CC-BY-SA

Notes:

- **Lecture L11: Communication Networks and Internet Protocols** 21.01.2025
- **Educational objective:** We cover foundations of telecommunication and introduce basic concepts of communication networks. We explain routing in packet-switched networks and cover basics of global Internet communication based on the TCP/IP protocol suite.
 - Foundations of Telecommunication
 - Packet-switched Networks and Routing
 - Local Area Networks
 - Internet Protocol Suite: IP, TCP and UDP

Notes:

Foundations of telecommunication

- ▶ telecommunication involves a **transfer information between two physical locations**
- ▶ we can use different **transmission media**

examples

- ▶ send **courier** via **roads**
 - ▶ send **container** via **pneumatic tube**
 - ▶ send **electrical current** via **wire**
 - ▶ send **light** via **space or optical fiber**
 - ▶ send **electromagnetic radiation** via **space**
-
- ▶ which of those can be used to **transfer binary data** between computers?



Ethernet cable

image credit: Wikimedia Commons, David.Monniaux, GNU license

Origins of electronic communication

- ▶ since 1830s: use of **electrical wires** for communication
- ▶ wires form **point-to-point connections** between **telegraph offices**
- ▶ 1860s: first **transatlantic telegraph line**
- ▶ how can we send information via an electrical wire?
- ▶ simplest idea: **on-off keying** to transmit digital code based on **presence/absence of electrical current** in a wire
- ▶ can you think of disadvantages of on-off keying?



International Morse Code	
<small>1: The length of a dot is one unit. 2: A dash is three units. 3: The space between parts of the same letter is one unit. 4: The space between letters is three units. 5: The space between words is seven units.</small>	
A	•••—
B	•••—•
C	—•••—
D	—•••
E	•••
F	•••—••
G	—••
H	••••
I	••
J	—•••
K	—••
L	••—•
M	—•—
N	••—
O	—•—•
P	••—••
Q	—•—••
R	••—••
S	•••••
T	—•
U	•••••
V	••••••
W	•••—
X	••—••
Y	—••••
Z	—•••••
1	•—••••
2	••—•••
3	•••—••
4	••••—•
5	•••••—
6	•••••••
7	••••••••
8	•••••••••
9	••••••••••
0	•••••••••••

Morse key (top) and
Morse code (bottom)

image credit: top: Hp.Baumeler, CC-BY-SA, bottom: public domain

Notes:

Notes:

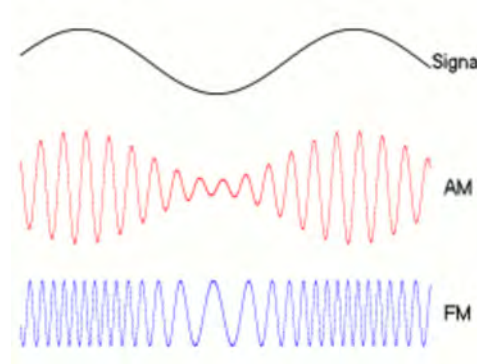
- A particularly easy way to communicate bits via an electrical wire is on-off keying. In this method, the presence of an electrical current represents a 1, while the absence of a current represents a 0. This is the basic idea behind the Morse code, which was used in early telegraphy.
- On-off keying also has a couple of disadvantages, namely that it requires a lot of power, that it is sensitive to noise (e.g. does absence of power mean 0 or a disruption of the line?), and that it is not possible to send multiple messages in parallel across a single line (i.e. no “multiplexing”).

Electromagnetic communication

- ▶ we can use **electromagnetic waves** for wire-based or wireless communication

terminology

- ▶ **waves** = oscillating change of medium
- ▶ **amplitude** = maximum oscillation
- ▶ **frequency** = oscillations per time unit
- ▶ alternating current in **transmitter antenna** generates **carrier wave**
- ▶ carrier wave induces alternating current in **receiver antenna**
- ▶ encode (analog or digital) signal by modulating **carrier wave**
- ▶ amplitude or frequency modulation (AM/FM)



Encoding of (analog) signal via Frequency or Amplitude Modulation of a Carrier Wave

image credit: Wikimedia user Berserkerus, CC-BY-SA
January 21, 2025

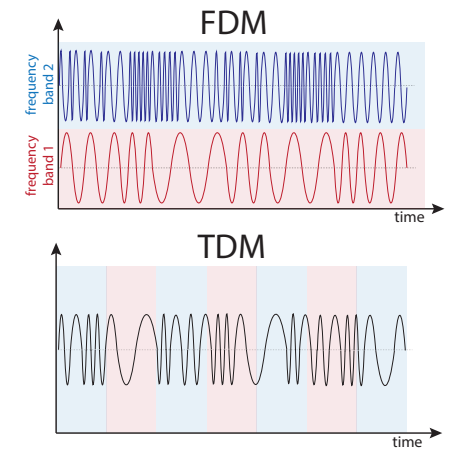
4

Multiplexing

- ▶ how can we **transfer multiple signals** via same medium (so-called “multiplexing”)?
- ▶ we can overlay multiple carrier waves in **different bands** (i.e. frequency-division multiplexing)
- ▶ **frequency-division multiplexing** can be used in wireless and wired communication

frequency-division multiplexing (FDM)

- ▶ broadcasting of multiple radio stations via **different radio frequencies**
- ▶ simultaneous transmission of multiple **TV channels** via single cable
- ▶ **alternative: time-division multiplexing (TDM)** assigns full frequency spectrum for certain **times**



frequency- vs. time-division multiplexing of two signals (red and blue)

Ingo Scholtes

Introduction to Informatics

Lecture 11: Communication Networks and Internet Protocol

January 21, 2025

5

Notes:

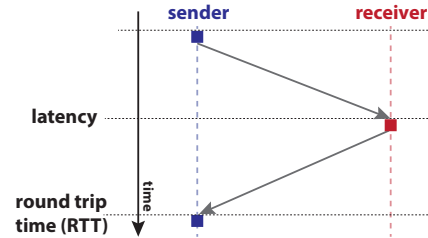
- To address these issues and to enable more efficient communication, we can use electromagnetic waves. These waves can be thought as oscillating changes in a medium (although in reality for electromagnetic waves there is actually mode medium), and they can be used to transmit information over long distances.
- How can we generate an electromagnetic wave? We can use a transmitter antenna, which is connected to an alternating current. This current generates a so-called electromagnetic carrier wave, which is then transmitted through the medium (e.g. a a wire, through the air or through the vacuum of space).
- A receiver antenna picks up the electromagnetic carrier wave and generates an alternating current in the receiver, which can be measured.

Notes:

- An important challenge (especially when we use wireless communication) is that we often need to transfer multiple signals over the same medium. This is where multiplexing comes into play. Multiplexing is the process of combining multiple signals into a single signal, which can then be transmitted over a single medium.

Latency and cosmic speed limit

- ▶ electromagnetic waves propagate at the **speed of light**, i.e. approx. 300,000 km/s
- ▶ **latency** is the time that is required by a signal to reach the destination
- ▶ **round-trip time (RTT)** is the time required to send a signal and to receive a response



exemplary round-trip times and distances

- ▶ **Würzburg – San Francisco** (approx. 9,200 km): 60 ms
- ▶ **low-earth orbit satellites (StarLink)** (approx. 300 – 500 km): 2 – 5 ms
- ▶ **geosynchronous orbit satellites** (approx. 72,000 km): 240 ms
- ▶ **Moon** (approx. 770,000 km): 2.5 s
- ▶ **Mars** (60 – 400 mio km): 6 – 45 mins

cosmic speed limit

based on Einstein's general relativity theory, no information can move faster than the speed of light

From telegraph to telephone

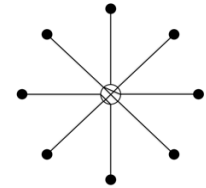
- ▶ 1876: Graham Bell presents **first telephone** that can transmit sound “telegraphically”
- ▶ early 1900s: millions of telephones
- ▶ how can we interconnect millions of communication endpoints?

endpoints vs. possible connections

for a communication network with n endpoints (nodes) there are $\frac{n \cdot (n-1)}{2}$ possible connections (links)

- ▶ 8 nodes → 28 connections
- ▶ 100 nodes → 4950 connections
- ▶ 10.000 nodes → 49.995.000 connections

- ▶ telephone providers used **circuit-switching** to establish direct line between communication partners



circuit-switched communication network with eight endpoints



telephone operators in 1952

image credit: Seattle Municipal Archives, CC BY 2.0

Notes:

Notes:

Computer communication via telephone?

- ▶ **idea:** use telephone lines for **point-to-point data transmissions**
- ▶ since 1920s: **teleprinters** for news wire services
- ▶ **modulator/demodulator device (modem)** allowed to transmit digital signals via telephone lines
- ▶ until 1990s, modems frequently used to exchange data via telephone network
- ▶ **transfer rate** limited to 56 KBit/s, i.e. approx. 7000 bytes per second
- ▶ **tele facsimile (telefax)** still widely used today (at least in Germany)

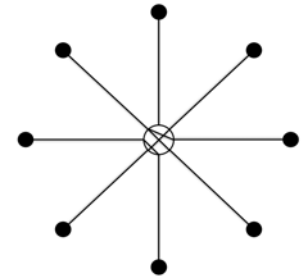


Modem (ca. 1960)

Notes:

Characteristics of circuit-switched networks

- ▶ before two endpoints can communicate we must **establish a connection**
- ▶ once connection is established, **line is exclusively available** to communication partners
- ▶ **centralized switch infrastructure** required to perform circuit-switching



advantages

- ▶ dedicated line allows to provide **guaranteed bandwidth, latency, etc.**
- ▶ direct communication between endpoints, i.e. **no intermediaries** that store/forward messages
- ▶ dedicated line facilitates **privacy of communication**

disadvantages

- ▶ overhead of **establishing connections** (e.g. short communication with many different endpoints?)
- ▶ exclusive use of lines, i.e. **inefficient use of available capacity**
- ▶ need for **centralized infrastructure** makes it prone to failure, sabotage, or attacks

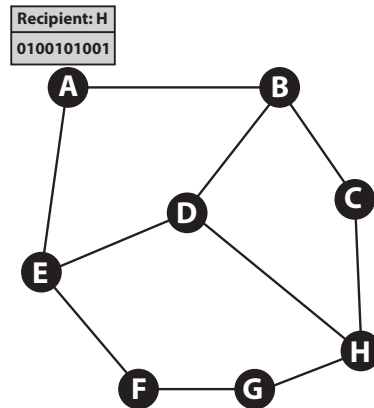
Notes:

From circuit- to packet-switching

- ▶ 1950s: fear of nuclear **attacks on communication infrastructure**
- ▶ 1960s: Paul Baran proposes decentralized **packet-switched communication network**
- ▶ hosts use direct connections to exchange **packets** (datagrams) consisting of **header and payload**
- ▶ header minimally contains **address of destination**

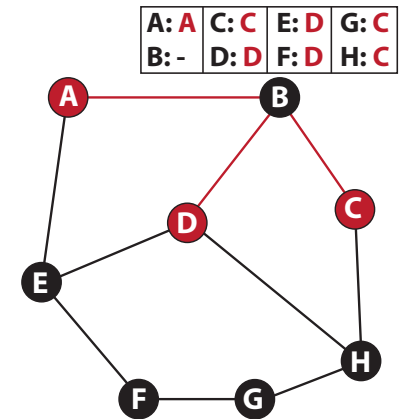
advantages of packet-switching

- ▶ communication lines **only occupied during transmission** of a packet
- ▶ **no need to establish connection** between sender and destination
- ▶ **no centralized circuit-switching infrastructure**



Routing packets

- ▶ in packet-switched networks there is **no direct line between sender and recipient**
- ▶ how can endpoints communicate if they are not directly connected?
- ▶ intermediate nodes **receive packets and forward them to the destination**
- ▶ **buffering of packets** if more packets arrive than can be forwarded immediately (store-and-forward)
- ▶ each host maintains a **routing table**
 - ▶ for given destination, table tells to which neighbour to forward a packet
 - ▶ routing tables (ideally) capture **shortest paths** in the network
 - ▶ tables can be updated when connections fail



Notes:

Notes:

Packet-switched wide area networks

- ▶ 1969: packet-switched military communication network **ARPANET** becomes operational
- ▶ 1976: **X.25 protocol** for packet-switched wide area networks (WANs)
 - ▶ specifies physical link technology, addresses, and packet format
 - ▶ provides “virtual call” abstraction that mimics direct line between endpoints

public X.25 data networks

- ▶ Telenet (USA, 1974)
 - ▶ DATAPAC (Canada, 1976)
 - ▶ DATEX-P (Germany, 1981)
 - ▶ AUSTPAC (Austria, 1982)
 - ▶ Minitel (France, 1982)
- ▶ modem used to connect home computer and device (DCE) that provides interface to X.25 network

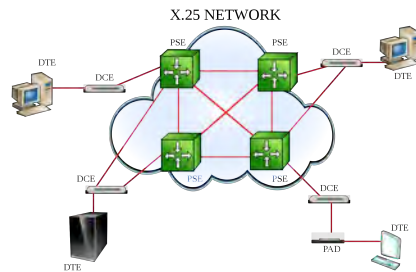


image credit: Wikimedia Commons, Adamantios, public domain

Packet-switched local area networks

- ▶ 1973: development of Ethernet standard for packet-switched **local area networks** (LAN)
- ▶ 1980: IEEE publishes IEEE 802 specification to standardize **Ethernet local area network**

modern Ethernet standards

- ▶ 1 GBit/s (approx. 120 MB/s) via copper cable
 - ▶ 10 GBit/s (approx. 1.2 GB/s) via copper cable
 - ▶ 100 GBit/s (approx. 10.2 GB/s) via optical fiber
 - ▶ 1.6 TBit/s (approx. 200 GB/s) via optical fiber
- ▶ Ethernet **is widely used** to interconnect computers at home or within companies



copper-based Ethernet cables with RJ45 connector

image credit: Wikimedia Commons, DiscDepotDundee.co.uk, CC-BY-SA

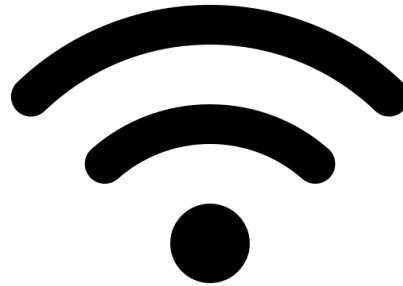
Notes:

- In X.25, the abbreviation DCE stands for Data Circuit-terminating Equipment, i.e. the device that connects a computer to the actual X.25 network - typically via a circuit-switched network like a telephone line. The counterpart at the user side is the DTE (Data Terminal Equipment).

Notes:

Wireless LAN

- ▶ 1997: development of IEEE 802.11 standard for **wireless local area networks** (WLAN)
- ▶ wireless communication via 2.4 GHz or 5 GHz electromagnetic carrier waves
- ▶ up to 100 meters distance (5 GHz) and up to 100 GBit/s (≈ 10 GB/s) (WiFi 8, Extremely High Throughput (EHT))
- ▶ **broadcast communication**, i.e. all station in vicinity receive transmission
- ▶ need to **encrypt communication** to maintain privacy
 - ▶ 1999: Wired Equivalent Privacy (WEP)
 - ▶ 2003: WiFi Protected Access (WPA)
 - ▶ WPA2 (2004) and WPA3 (2018)
- ▶ apart from communication, WLAN can be used for **geolocation** and **WiFi sensing**



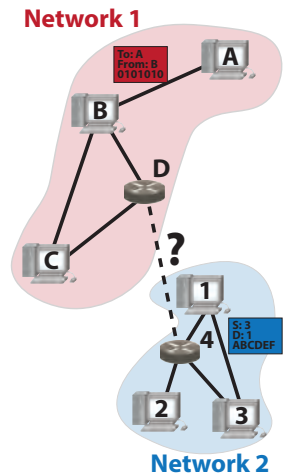
WiFi symbol

image credit: Font Awesome, CC-BY-SA

Notes:

Internet Communication

- ▶ since 1970s: many packet-switched communication networks operated by different providers
 - ▶ different packet formats and communication protocols
 - ▶ different addressing schemes
 - ▶ different routing protocols
- ▶ problem: can we enable **packet-switched communication across these different networks**?
- ▶ need a way to packetize data, route packets, and address computers **across organizational domains**
- ▶ 1974: first version of **Internet Protocol (IP)** for inter-network communication by Vinton Cerf and Robert Kahn
- ▶ 1975: interconnection between computer networks in Stanford and London is **first seed of the "Internet"**



Notes:

Internet Protocol (IP)

- ▶ **basic idea**: add **Internet protocol layer** for communication across networks
- ▶ add additional **IP header** to payload of packets
- ▶ to route packets we need unified way to **address destination network** and to **address destination computer** (=host)
- ▶ IP address = numerical address with **variable network and host component**
- ▶ **IPv4 address** = four groups of 8 bits = 32 bits, i.e. address space of approx. 4 billion addresses
- ▶ **ICANN assigns IP addresses** to organizations
- ▶ **IPv6 address** = eight groups of 16 bits = 128 bits, i.e. address space of approx. 10^{38} addresses

IPv4 address of a computer on the Internet

132.178.253 . 17
network host

special IPv4 addresses

- ▶ 127.0.0.0 – 127.255.255.255: **loopback address** for diagnostic purposes
- ▶ 192.168.0.0 – 192.168.255.255: **addresses for private networks**, no routing on Internet
- ▶ 255.255.255.255: **broadcast address**

What is an IP packet?

- ▶ thanks to additional **IP header** we can exchange **IP packets** across networks
- ▶ header contains **IP address of source and destination**
- ▶ to send IP packet across specific link, we wrap it into packet defined by **link-layer protocol** (e.g. Ethernet)
- ▶ link-layer protocol can use its own addressing scheme to address devices
- ▶ example: Ethernet
 - ▶ Ethernet packets are used to transport IP packets between devices in the same network
 - ▶ **Ethernet MAC address** = regionally unique 48 bit hardware address
 - ▶ MAC address identifies specific device in a local network

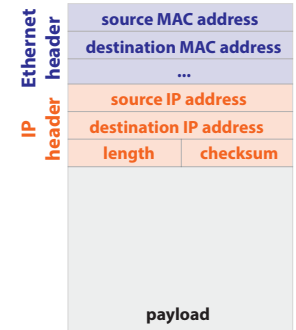


image credit: Raimond Spekking / CC BY-SA 4.0

Notes:

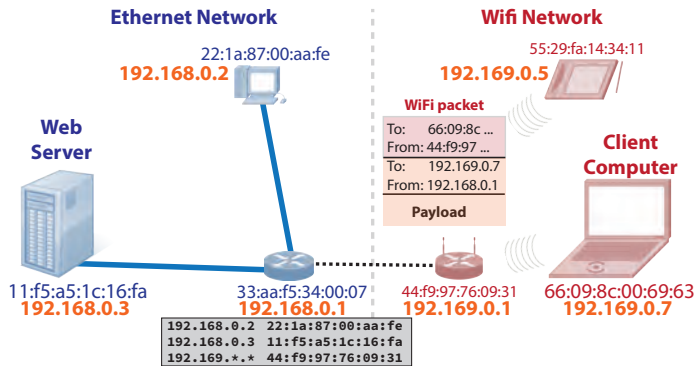
- ICANN = Internet Corporation for Assigned Names and Numbers

Notes:

- We can think of the following analogy for the relationship between IP packets and the link-layer protocol:
 - we can write a letter (this is the payload) and put it in an envelope that contains the IP address of the sender and the recipient (this is the IP header). This allows us to send our payload to a recipient that can be in any network in the world.
 - Assuming that we cannot send the letter directly, because the recipient is not in our network, we now put this whole envelope into another envelope, which contains the address of the sender and a recipient that is able to forward the letter to us (this outer envelope is the link-layer packet, which contains the IP packet).
 - We can now send this (outer envelope) via a postal system that understands the address format of the outer envelope (this is link-layer protocol). The recipient opens the envelope and finds the inner envelope with the IP packet. Using the IP address of the inner envelope it then determines to whom to forward the letter via a direct link. For this, it again wraps the inner envelope with our letter (i.e. the IP packet) into a new envelope that is again sent to the next hop via the link-layer protocol.

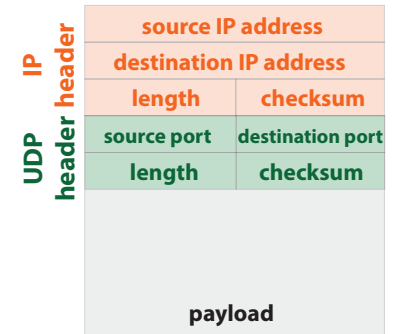
Schematic view of Internet routing

- ▶ **link-layer protocol** is used to exchange IP packets computers that are **directly connected (e.g. in the same local Ethernet network)**
- ▶ **IP addresses** of IP packet is then used to route packages across networks



User Datagram Protocol (UDP)

- ▶ IP is a **host-to-host protocol**, i.e. allows to exchange packets between **computers**
- ▶ IP address identifies destination computer (=“host”) on the Internet
- ▶ **but:** one computer can simultaneously “host” **multiple applications** (e.g. WhatsApp client, Web browser, YouTube client, etc.) that all want to communicate
- ▶ how can we send a packet to a **specific application** (e.g. web server or browser) running on a host computer?
- ▶ need **transport-layer protocol** that minimally handles addressing of processes/applications on a machine
- ▶ UDP can be used to send packets (datagram) to specific processes identified by **16-bit port number**



exemplary **UDP endpoint address**

192.168.0.1 : 53
 network host port

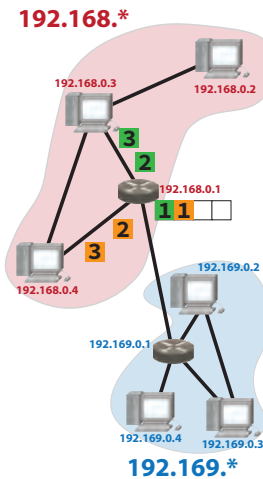
Notes:

- This general idea of routing IP packets across networks is shown in the schematic view above. Here we assume that the hosts are in two different networks, each having its own addressing scheme and packet format. Two routers connect those two networks and each router has (a) direct connections to the computers in one of the networks, and (b) a direct connection to the router in the other network.
- We consider the situation that the Web server (left) wants to send a packet to the Client computer (right). The Web server create an IP packet, setting the address 192.169.0.7 as destination and wrapping the packet into an Ethernet packet. It then sends the Ethernet packet with the contained IP packet to the router. The router receives the packet, unwraps the IP packet and checks the IP address of the destination. It determines the destination network and finds that it has a direct link to the router connecting to this network. It then wraps the IP packet into another link-layer packet that is sent to the router in the other network. This router then eventually forwards the IP packet to the Web browser (wrapped in another link-layer packet).

Notes:

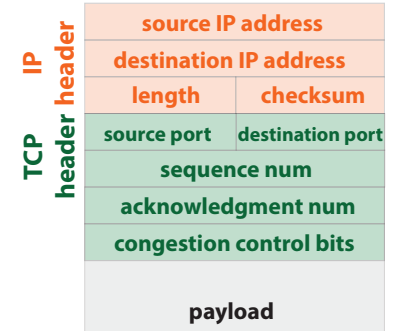
Challenges of datagram communication

- ▶ UDP packet is called **datagram**, i.e. **independent packet** that can be sent across networks
- ▶ on their path to the destination host, **packets may need to be buffered** by intermediate routers
- ▶ in case of **network congestion**, routers may need to drop packets stored in the (limited-size) buffer storage
- ▶ Ethernet, IP, and UDP provide **no guarantee that packet reaches destination**
- ▶ consecutive packets can take different routes, i.e. **UDP packets may be received out-of-order**
- ▶ how can we solve these challenges of **datagram communication**?



Transmission Control Protocol (TCP)

- ▶ instead of UDP, we can use the **connection-oriented transport-layer protocol TCP**
- ▶ to guarantee (in-order) delivery of packets, TCP endpoints first **establish (virtual) connection**
- ▶ **sequence number** of TCP header is used to **provide in-order delivery of packets**
- ▶ to guarantee delivery, destination **acknowledges receipt of packages**
- ▶ but: resending of packets is problematic if packets are lost because intermediate buffers are full ("**congestion**")
- ▶ **congestion control mechanism** is used to automatically adjust transfer speed to network capacity



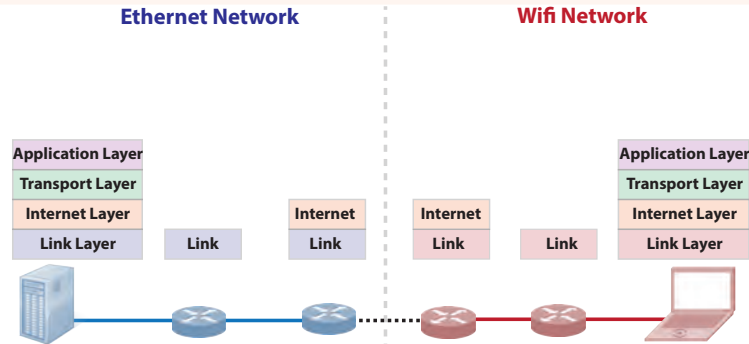
Notes:

Notes:

End-to-end principle and protocol layers

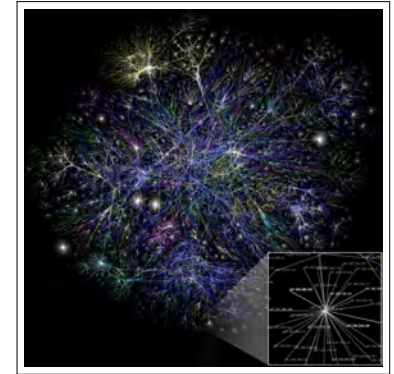
end-to-end principle

Intermediate computers or routers only provide best-effort forwarding of packets. Further guarantees (e.g. in terms of reliable communication, security, authenticity, etc.) must be implemented in **communication endpoints**. → JH Saltzer, DP Reed, DD Clark, 1984



Conclusion

- ▶ we covered basics of **telecommunication** and introduced **packet-switched networks and routing**
- ▶ we discussed key **Internet protocols** like IP (Internet layer), UDP, and TCP (both transport layer)
 - ▶ **no central authority** that manages Internet infrastructure
 - ▶ **end-to-end design principle** simplifies design of Internet routers and protocols
 - ▶ **net neutrality** = Internet infrastructure (i.e. routers) should not care **which applications communicate**
 - ▶ **security and privacy challenges** must be addressed by **application-layer protocols**



partial map of the Internet as of January 15, 2005
each node is one computer network

image credit: Opte project, CC-BY-SA

Notes:

- The figure above gives another schematic view of Internet communication in our example with two interconnected networks from before. We can actually structure the protocols that are involved in the communication between endpoints into multiple layers.
- At the lowest layer, we have considered link-layer communication protocols like Ethernet or Wifi, which just handle the communication between endpoints that are directly connected in the same (local) network. Each network can use its own link-layer protocol, e.g. in the example above the cable-based Ethernet protocol or the wireless protocol WiFi.
- On top of this, we have the Internet layer, which is responsible for routing packets across networks. The Internet layer is implemented by the Internet Protocol (IP). IP packets are used to send data between computers that are not directly connected in the same network. The IP packets contain the IP address of the sender and the recipient, which allows routers to forward the packets across networks. From the devices shown above, the endpoints as well as the routers that connect different networks must implement the Internet Protocol. Intermediate devices that are only responsible for forwarding packets within a network do not need to implement the Internet Protocol.
- On top of the Internet layer, we have the transport layer, which is responsible for addressing processes on a computer. The transport layer is implemented by the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP is a connection-oriented protocol that guarantees reliable and in-order delivery of packets. UDP is a datagram protocol that does not guarantee the (in-sequence) delivery of packets. Both protocols use port numbers to address processes on a computer.

Notes:

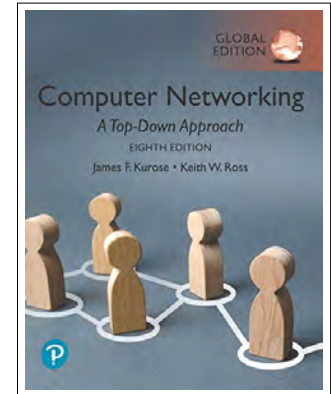
Self-study questions

1. Explain how we can use electromagnetic waves to transfer a signal.
2. How can multiple signals be transmitted simultaneously via a single wire?
3. What are the advantages and disadvantages of circuit- and packet-switched networks?
4. Consider a communication network with 1000 endpoints. How many bidirectional connections are possible between those endpoints?
5. Explain how packets in a packet-switched network are routed between sender and destination.
6. What is Ethernet and what is it used for?
7. Which additional privacy challenges arise in WLAN compared to Ethernet.
8. What is the difference between a MAC and IP address?
9. Outline the structure of an IP packet.
10. How are packets between two networks routed in the Internet?
11. What is the end-to-end principle?
12. Give an example for a situation in which an IP packet may not arrive at the destination.
13. What are advantages and disadvantages of UDP and TCP?
14. What is the difference between a link-layer, Internet, transport, and application-layer protocol?

Literature

reading list

- ▶ Paul Baran: **On Distributed Communications**, RAND Memorandum, 1964
→ https://www.rand.org/pubs/research_memoranda/RM3420.html
- ▶ Robert Metcalfe: **Packet Communication**, MIT Technical Report MAC TR-114, 1973
→ <https://dspace.mit.edu/handle/1721.1/5739>
- ▶ V Cerf, R Kahn: **A Protocol for Packet Network Intercommunication**, IEEE Transactions on Communications, 1974
→ <https://www.cs.princeton.edu/courses/archive/fall120/cos461/papers/TCPIP74.pdf>
- ▶ JH Saltzer, DP Reed, DD Clark: **End-to-end arguments in system design**, ACM Transactions on Computer Systems, 1984
→ <https://dl.acm.org/doi/10.1145/357401.357402>
- ▶ Information Sciences Institute: **Transmission Control Protocol**, IETF RFC 793, 1981
→ <https://tools.ietf.org/html/rfc793>



Notes:

Notes:

Introduction to Informatics for Students from all Faculties

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

Motivation

- ▶ we introduced **TCP/IP protocol suite** for communication across packet-switched networks
- ▶ Internet builds on **protocol stack** that consists of link-, network, transport-, and application layer protocols

Internet protocol stack

- ▶ **link layer**: direct communication between **network devices** within one network (e.g. Ethernet)
- ▶ **network layer**: addressing and routing between **hosts** in multiple networks (IP)
- ▶ **transport layer**: packet- or connection-oriented communication between **processes running on ports**

- ▶ Which **Internet-based applications** did you use today?

Application Layer

Transport Layer

Internet Layer

Link Layer

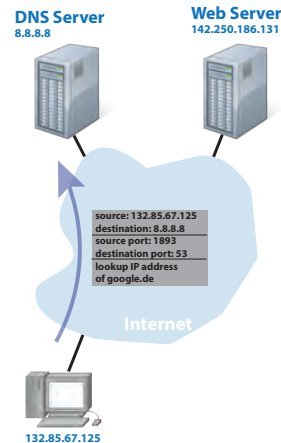
Notes:

- **Lecture L12: Internet Applications and WWW** 28.01.2025
- **Educational objective:** We introduce technical foundations of the World Wide Web and explore basic technologies like HTTP, HTML, CSS, and JavaScript. We highlight the importance of Web standards for modern Cloud Computing Services.
 - Security and Privacy Issues in the Internet
 - World Wide Web and HTTP
 - HTML, CSS, and JavaScript
 - Cookies, Web Services, and Cloud Computing

Notes:

Domain Name System (DNS)

- ▶ similar to phone numbers, **IP addresses are not human-friendly**
- ▶ need global directory service that allows to **map human-friendly hostnames to machine-friendly IP address**
- ▶ **Domain Name System** = hierarchy of **name servers** to resolve hostname to IP address
- ▶ **DNS protocol** specifies format of DNS request/response messages
- ▶ DNS requests can be sent via UDP (or TCP) to port number 53
- ▶ hierarchical naming scheme organized into different **top-level domains** (e.g. .com, .de)
- ▶ domain names **managed by ICANN**

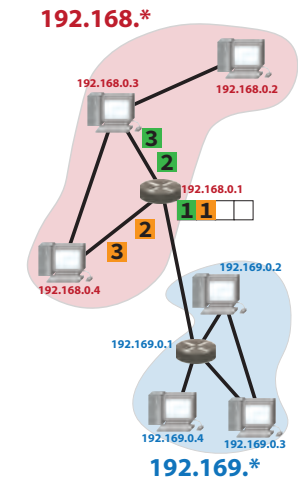


Security and Privacy Issues

- ▶ decentralized design of the Internet introduces numerous **security and privacy challenges**
- ▶ is this really a design flaw of the Internet?

fundamental challenges

- ▶ **simplicity** of core Internet protocols is recipe for their success
- ▶ **end-to-end principle**: security or privacy guarantees must be provided at endpoints
- ▶ intermediaries not under central control, i.e. we cannot trust them



Notes:

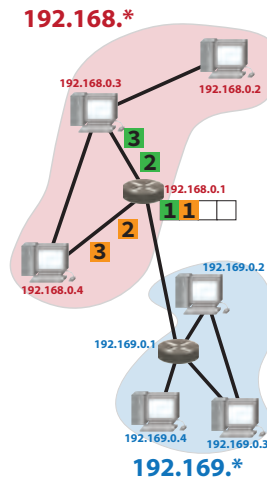
Notes:

Privacy and User Tracking

- ▶ packets sent between endpoints can be **intercepted by intermediate routers** (or other machines with access to the network)
- ▶ we can add **application-level encryption** of payload data
- ▶ **Transport-Layer Security (TLS)** can be used to encrypt communication for different application-level protocols
- ▶ but: **unencrypted meta-data** (e.g. IP address, ports) allow to **track users**

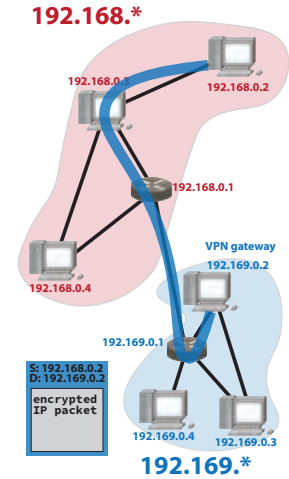
privacy-enhancing techniques

- ▶ private relays (e.g. Apple iCloud)
- ▶ MAC randomization
- ▶ IPv6 privacy extensions
- ▶ virtual private networks (VPN)



Virtual Private Networks (VPN)

- ▶ using, e.g. TLS, we can establish **encrypted connection** between endpoints
- ▶ rather than publicly routing IP packets, we can **use secure tunnel to send packets to gateway** in a remote network
- ▶ idea: wrap IP packet into encrypted payload of packets sent to gateway
- ▶ allows to **securely connect two private networks via a public network**, e.g. different company sites
- ▶ VPN clients can be used for **secure remote access to restricted network resources**
- ▶ public VPN services to **hide IP address from Internet provider**, circumvent geo-blocking or censorship



Notes:

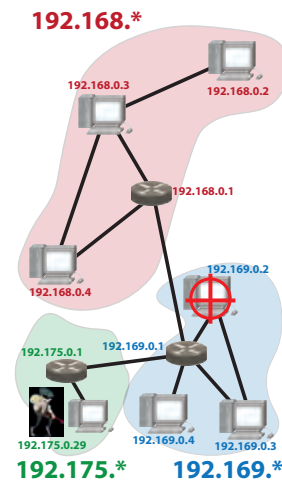
Notes:

Hacking and Firewalls

- ▶ hackers **exploit security vulnerability** to gain access to service, computer, network

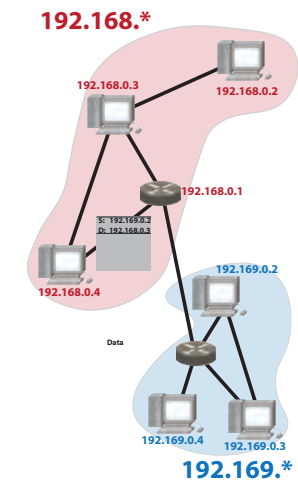
hacking terminology

- ▶ **exploit** = code that uses a known security vulnerability to perform an attack
- ▶ **zero-day exploit** = exploit that uses previously unknown vulnerability for which no patch is available
- ▶ **white-hat hacker** = person that uses hacking techniques to identify (and fix) security issues
- ▶ **firewall** allows to restrict communication to/from machine or network
- ▶ set of rules to **selectively allow or disallow communication** to/from certain ports, hosts, networks, countries, etc.



IP spoofing

- ▶ upon sending of an IP packet, **machines add their own source address to IP packet**
- ▶ correctness of this address is **generally not verified**
- ▶ forging of wrong source address is called **IP spoofing**
- ▶ how can we avoid IP spoofing?
 - ▶ router can ensure that no **incoming packet** has source address in its own network
 - ▶ router can ensure that no **outgoing packet** has sender address outside its own network

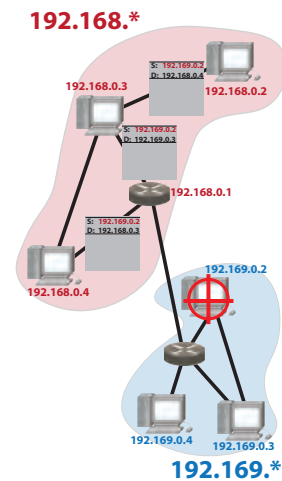


Notes:

Notes:

Denial-of-Service attacks

- ▶ **denial-of-service (DoS)** attack aims at flooding target with superfluous packets
- ▶ DoS attack can **disrupt service** (often combined with blackmailing of provider)
- ▶ **distributed-reflected-denial-of-service attack** is based on IP spoofing
- ▶ attackers send **requests with spoofed source address** to large number of random computers
- ▶ responses sent to spoofed source address are flooding the target
- ▶ use of IP spoofing makes it **hard to trace back attack**

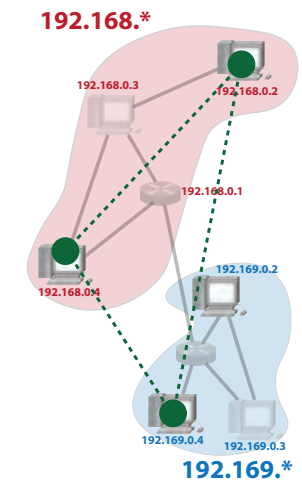


P2P Overlay Networks

- ▶ we can use IP-based communication between hosts to establish **Peer-to-Peer (P2P) overlay networks**
- ▶ application-layer encryption allows to establish **secure channels** between peers
- ▶ messages can be **(anonymously) routed within P2P network** via overlay links based on custom communication protocols

exemplary P2P overlay networks

- ▶ **Gnutella** Filesharing Platform
- ▶ **Bitcoin** distributed ledger
- ▶ **Tor** for anonymous communication



Notes:

Notes:

Tor and Darknet

- ▶ terms **Darknet and Darkweb** collectively refer to overlay networks that allow fully anonymous and secure communication

example: Onion routing

- ▶ P2P-based routing of messages via **multiple encrypted protocol layers**
- ▶ basis of The Onion Router (Tor)
- ▶ no central servers, i.e. P2P services are difficult to take down
- ▶ important for **whistleblowers and resistance of censorship**
- ▶ but: often used for **illegal activities** (e.g. SilkRoad market)

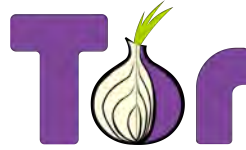


image credit: Wikimedia, M4sugared (bottom), The Tor Project (top), CC-BY-SA January 28, 2025

History of the World Wide Web (WWW)

- ▶ since late 1980s: global system of **interconnected networks** based on TCP/IP
- ▶ at first **predominantly used in academia**, private use limited to few enthusiasts
- ▶ how can we **use Internet to share information** (e.g. research results)?
- ▶ in 1990, Tim Berners-Lee develops prototype of **World Wide Web**
- ▶ World Wide Web quickly spreads at CERN (and beyond)
- ▶ 1995: with NetScape Navigator, WWW becomes **global phenomenon** → browser wars: 1995 – 2001
- ▶ 2000: burst of **dot-com bubble**



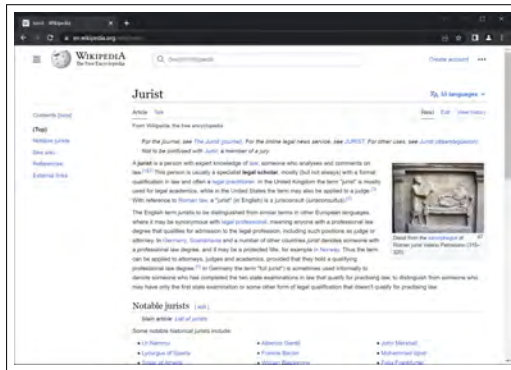
Tim Berners-Lee

image credit: CERN, CC-BY-SA (bottom), Robert Cailliau (top)

Notes:

Notes:

WWW - A user perspective



questions

- ▶ How can we **publish and retrieve Web pages** on the Internet?
- ▶ How are Web pages transferred between **Web servers** and **Web browsers**?
- ▶ How can we develop **interactive Web-based applications**?

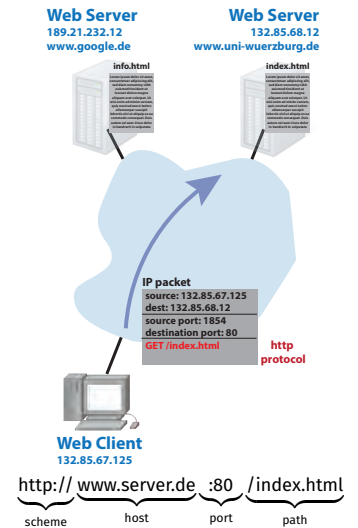
Foundations of the WWW

- ▶ WWW = **Web resources** (e.g. **hypertext documents, images, data, code, services**) hosted on millions of **Web Servers**
- ▶ Web resource can be located via **Uniform Resource Locator (URL)**

URL components

- ▶ **scheme** defines application-layer protocol to access resource
- ▶ **host** identifies DNS hostname or IP of host machine
- ▶ optional **port number** identifies application
- ▶ **path** locates resource on the host (e.g. specific service or file)
- ▶ optional **query string** can be passed to resource

- ▶ **Web client** uses **hypertext transfer protocol (HTTP)** to interact with Web resources



Notes:

Notes:

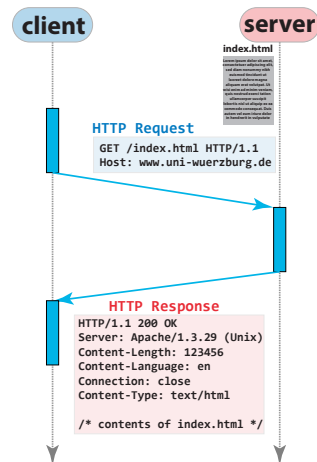
Hypertext Transfer Protocol

- ▶ clients use **hypertext transfer protocol (HTTP)** to interact with Web server

- ▶ simple **request-response protocol**

process

- ▶ HTTP server waits for requests (e.g. on **standard port 80**)
 - ▶ HTTP client sends **HTTP Request** message to server
 - ▶ server accesses **resource** (e.g. web page, database, etc.)
 - ▶ server sends **HTTP Response** with representation of requested resource to client
-
- ▶ one of the foundations of the World Wide Web (and cloud computing)
 - ▶ first proposed by Tim Berners-Lee in 1989



HTTP request methods

- ▶ HTTP defines different **request methods**

HTTP Request Methods (excerpt)

- ▶ **HTTP GET**: retrieve resource
- ▶ **HTTP POST**: process resource in request
- ▶ **HTTP PUT**: create or update resource
- ▶ **HTTP DELETE**: delete resource

- ▶ with GET method, client requests **resource identified by URL**

- ▶ example for resource: file or data on server
- ▶ client can specify **accepted representations** of resource

- ▶ **request parameters** can be passed using **query part** of URL

- ▶ HTTP can be used to access **Web services**, e.g. get specific record from DB in JSON format

HTTP GET request by client

```
GET /db?student=42 HTTP/1.1
Host: db.uni-wuerzburg.de
Accept: application/json
```

HTTP response by server

```
HTTP/1.1 200 OK
Server: Apache/1.3.29
Content-Length: 1232
Content-Language: en
Connection: close
Content-Type: text/html
```

/* representation of /x */

Notes:

Notes:

Hypertext Markup Language (HTML)

- ▶ HTTP = application-layer protocol to **transfer resources** via TCP/IP
- ▶ HTTP is frequently used to transfer **hypertext documents**
- ▶ **Hyper-Text Markup Language (HTML)**
 - ▶ standard document format for **Web pages**
 - ▶ introduced in 1993
- ▶ HTML = XML-based **markup language** designed to **render Web pages**
- ▶ markup = annotations that define the semantic structure of a document
- ▶ **declarative language**, i.e. we describe structure of Web page, not how it is rendered

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>ACME Vacuum Cleaner</title>
</head>
<body>
  <div id="productInfo">
    <h1>ACME Vacuum Cleaner</h1>
    
    <br />
    <ul>
      <li>Model: ACME 1400</li>
      <li>Power consumption: 1400 W</li>
    </ul>
  </div>
  <div id="stockInfo">
    <ul>
      <li>Price: 149.99 Euro</li>
      <li>Availability: in stock</li>
    </ul>
  </div>
</body>
</html>
```

HTML basics

- ▶ **HTML header** contains **information about the document**
 - ▶ title of page (<title>)
 - ▶ meta information (<meta>)
 - ▶ script files (<script>)
 - ▶ visual styles (<style>)
 - ▶ related files (<link>)
- ▶ **HTML body** contains **semantically structured document contents**
 - ▶ document divisions (<div>)
 - ▶ headings (<h1> - <h6>)
 - ▶ media (, <video>, <audio>)
 - ▶ text paragraphs (<p>)
 - ▶ anchors for hyperlinks (<a> element)
 - ▶ (ordered) lists (and)
- ▶ hyperlinks allow to **reference other Web pages on any Web server based on their URL**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <script> ... </script>
  <style> ... </style>
  <link rel="stylesheet" type="text/css" href="example.css" />
  <title>ACME Vacuum Cleaner</title>
</head>
<body>
  <div id="productInfo">
    <h1>ACME Vacuum Cleaner</h1>
    
    <br>
    <ul>
      <li>Model: ACME 1400</li>
      <li>Power consumption: 1400 W</li>
    </ul>
    <p>This vacuum cleaner ... </p>
  </div>
  ...
</body>
</html>
```

Notes:

Notes:

HTML forms

- ▶ forms allow to **collect user input**
 - ▶ single-line input (<input>)
 - ▶ multi-line text boxes (<textarea>)
 - ▶ lists and options (<select>, <option>, <datalist>)
- ▶ type of single-line input can be chosen via **type attribute**
 - ▶ text, password, email, ...
 - ▶ number, range, date, time, ...
 - ▶ radio button or checkbox
 - ▶ form buttons
- ▶ clickable **button elements with arbitrary content**
- ▶ **client sends contents of form fields via HTTP GET or HTTP post**

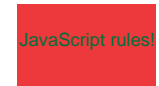
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>ACME Vacuum Cleaner</title>
  </head>
  <body>
    <select>
      <option value="1400">1400</option>
      <option value="20k">20k</option>
    </select>
    <div id="productInfo">
      <h1>ACME Vacuum Cleaner</h1>
      <ul>
        <li>Model: ACME 1400</li>
        <li>Power consumption: 1400 W</li>
      </ul>
    </div>
    <div id="stockInfo">
      <ul>
        <li>Price: 149.99 Euro</li>
      </ul>
      <form action="order" method="GET">
        <input type="number" name="number" />
        <input type="submit" value="Buy" />
      </form>
    </div>
  </body>
</html>
```

```
...
<form action="order" method="GET">
  <input type="number" name="number" />
```

Multimedia contents

- ▶ images can be embedded via
- ▶ video and audio playback via <audio> and <video>
 - ▶ since HTML5: **no need for plugins like FlashPlayer**
 - ▶ mp4 video and mp3 audio supported by all browsers
- ▶ interactive 2D and 3D visualizations via <canvas>
 - ▶ drawing of 2D shapes by scripts (e.g. JavaScript)
 - ▶ web-based 3D rendering via WebGL

```
<!DOCTYPE html>
<html>
<body>
  <canvas id="myCanvas">
    Your browser does not support the
    HTML5 canvas tag.
  </canvas>
  <script>
    var c = document.getElementById("myCanvas");
    var ctx = c.getContext("2d");
    ctx.fillStyle = "#FF0000";
    ctx.fillRect(0, 0, 170, 100);
    ctx.font = "30px Arial";
    ctx.fillStyle = "#00FFAA";
    ctx.fillText("JavaScript rules!", 10, 50);
  </script>
</body>
</html>
```

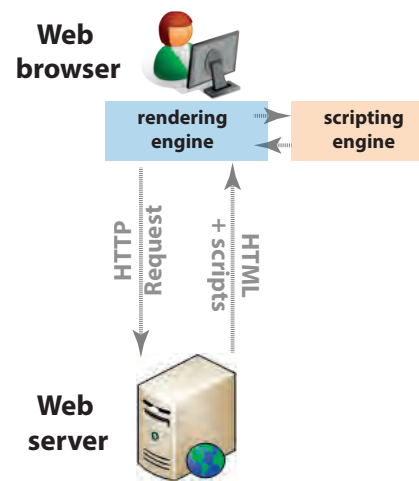


Notes:

Notes:

Server- and Client-side Scripts

- ▶ WWW is more than a collection of **static Web pages**
- ▶ use of **server-side scripts** (e.g. python, C++, PHP) run by Web server allows to **programmatically generate Web pages**
- ▶ but: any update of currently displayed page would require that ...
 1. HTTP request must be sent to server
 2. new page in HTTP response must be rendered by browser
- ▶ to provide **interactive Web applications** we can use **client-side script** that runs in the browser
- ▶ client-side script **manipulates currently displayed Web page**



JavaScript

- ▶ JavaScript = most popular client-side scripting language
- ▶ **object-oriented and interpreted programming language**
- ▶ syntax **influenced by Java**, but otherwise unrelated
- ▶ we can use document object to **access and manipulate structure of currently displayed HTML document**
- ▶ **JavaScript Object Notation (JSON)** has become a standard text-based format to exchange data on the Web

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <script src="script.js"
    type="text/javascript" />
  <title>Title</title>
</head>
<body>
  <button onclick="addListElem();">
    Click me!
  </button>
  <ul>
    <li>First element</li>
  </ul>
</body>
</html>
```

```
function addListElem() {
  var ul = document.
    getElementsByTagName('ul')[0];
  var new_li = document.createElement('li');
  var text = document.
    createTextNode('Second element');
  new_li.appendChild(text);
  ul.appendChild(new_li);
}
```

Notes:

Notes:

HTML integration of JavaScript

- ▶ JavaScript code can be provided ...
 1. **inline** within **event attributes** (e.g. onClick or onHover) of HTML elements
 2. within script element inside the **HTML header**
 3. in a **separate script file** that is referenced via script element
- ▶ scripts are executed in **sandbox**, i.e. no access to local filesystem but access to **isolated local storage**

Same-Origin-Policy

- ▶ applications can only access data that have the same **origin** defined as protocol + host + port
- ▶ crucial to ensure data integrity and security on the Web
- ▶ example: script on random website cannot access data stored by website of your bank

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <script src="script.js"
    type="text/javascript" />
  <title>Title</title>
</head>
<body>
  <button onclick="greet('students');">
    Click me!
  </button>
</body>
</html>
```

```
function greet(name) {
  alert('Hello' + name);
}
```

Cookies and user tracking

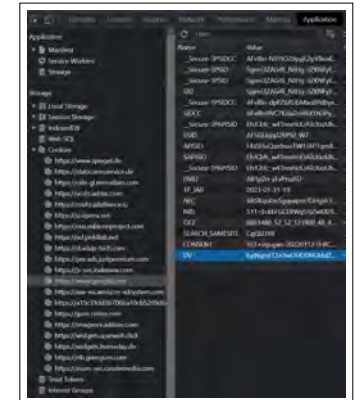
how can Web page memorize data across sessions?

- ▶ contents of shopping cart in an online shop (when you are not logged in)
- ▶ information on user identity when visiting online shop

- ▶ Web pages must be able to **store information on a user's machine**
- ▶ cookie = small textfile with key-value data, created by Web server or JavaScript
- ▶ **tracking cookies** can be used to track users across websites

examples

- ▶ advertisement banners included by multiple websites
- ▶ Google analytics tracking code



Notes:

Notes:

Summary

- ▶ we covered technological foundations of the **World Wide Web**
- ▶ WWW = suite of application-layer protocols (esp. HTTP) and standard data formats (esp. HTML) exchanged via TCP/IP
- ▶ **Web is more than just Web pages and browsers!**
- ▶ TCP/IP + Web technologies = **infrastructure for globally distributed applications** that communicate via standard protocols and formats



Self-Study Questions

- ▶ Explain how names are resolved to IP addresses in the Internet.
- ▶ What is IP spoofing and how can it be avoided?
- ▶ List five security and privacy challenges that emerge due to the Internet's decentralized design and explain how we can address them.
- ▶ What is a Virtual Private Network (VPN)?
- ▶ Discuss to what extent a VPN facilitates anonymous communication on the Internet.
- ▶ What is a Firewall?
- ▶ Explain the role of IP spoofing in Denial-of-Service attacks.
- ▶ What do the terms Darknet and Darkweb refer to?
- ▶ Explain the relationship between the Internet and the World Wide Web?
- ▶ Explain the components of a Uniform Resource Locator.
- ▶ Explain the process behind a typical Web request.
- ▶ What are cookies and how do they affect the privacy of Web users?
- ▶ What is the Same-Origin-Policy and what is a Cross-Site-Scripting attack?

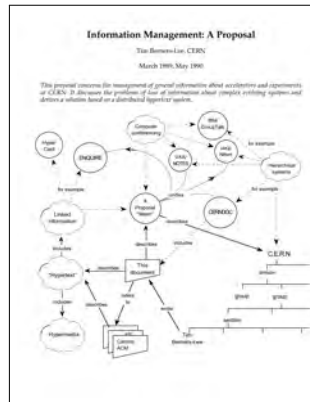
Notes:

Notes:

Literature

reading list

- ▶ T Berners-Lee: **Information Management: A Proposal**, CERN, 1990
→ <http://www.w3.org/History/1989/proposal.html>
- ▶ T Berners-Lee: **The Original HTTP as defined in 1991**, w3.org
→ <http://www.w3.org/Protocols/HTTP/AsImplemented.html>
- ▶ Self-HTML Tutorial: **HTML, CSS, JavaScript**, selfhtml.org
→ <https://wiki.selfhtml.org/>
- ▶ W3.org: **Same Origin Policy - Web Security**
→ https://www.w3.org/Security/wiki/Same_Origin_Policy



Notes: