MSc Industrial Automation - System Design

Prof. Dr.-Ing. Werner Zimmermann

I. System Development Process

- Definition of a system
- Decision making:
  - definition of goals
  - Eisenhower principle
  - weighted criteria assessment
- Planning:
  - activity lists
  - time schedule
- Development process
  - top down, bottom up
  - partitioning and interfaces
- Phase models
  - actions, results, milestones
- Requirements analysis

II. Analytical System Design

- Modeling dynamic systems
  - state variables
  - describing equations
  - block diagrams
- State space control
  - state space description
  - control structure
  - pole placement design
- Observers
  - observer structure
  - pole placement design
- Optimization of control systems
- Identification of control plants
- Neuronal Networks
  - application in control systems

Module System Development Process

Contents

Index

Module System Development Process

A. Overview

1. Technical systems

A car as a technical system:

A system

- consists of **components**
  e.g. engine, brakes, ...
- performs **functions**
  e.g. accelerating, braking, ...
- has **external interfaces** (to the outside world/environment)
  e.g. accel. and brake pedal, tires, ...

  and **internal interfaces** (between the components)
  e.g. accelerator pedal - engine

Most systems

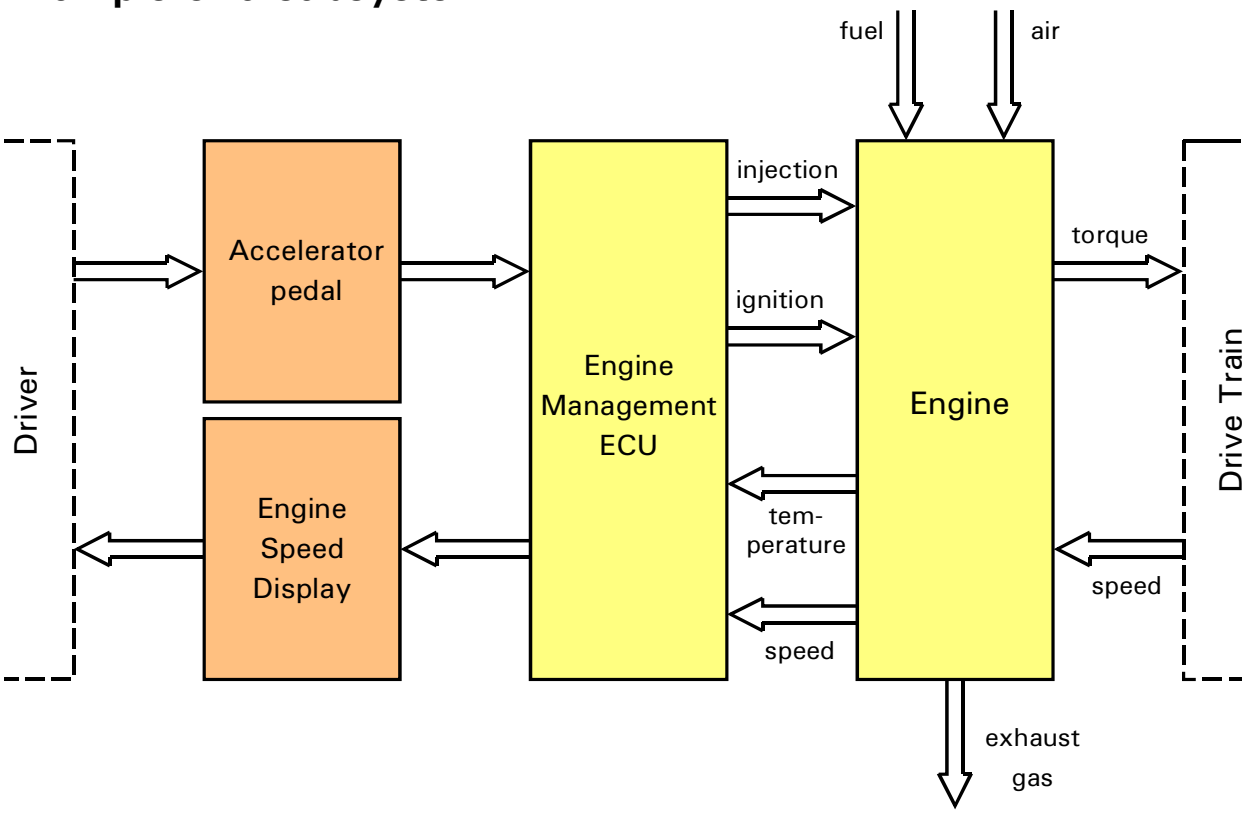
- are components of **super-systems**, i.e. higher level system

  e.g. super-systems of a car:
  - a car is a component of the system 'traffic'
  - 'traffic' is a component of the system 'city'
  - . . .

- can be subdivided in **subsystems**, i.e. lower level systems

  e.g. subsystems of a car:
  - engine and engine management system
  - brakes and brake management system (ABS/ASR)
  - . . .

Example of a subsystem



System border

- defines what is considered to be inside and outside of the system
- can be arbitrarily chosen by the system engineer depending on what is considered to be of interest

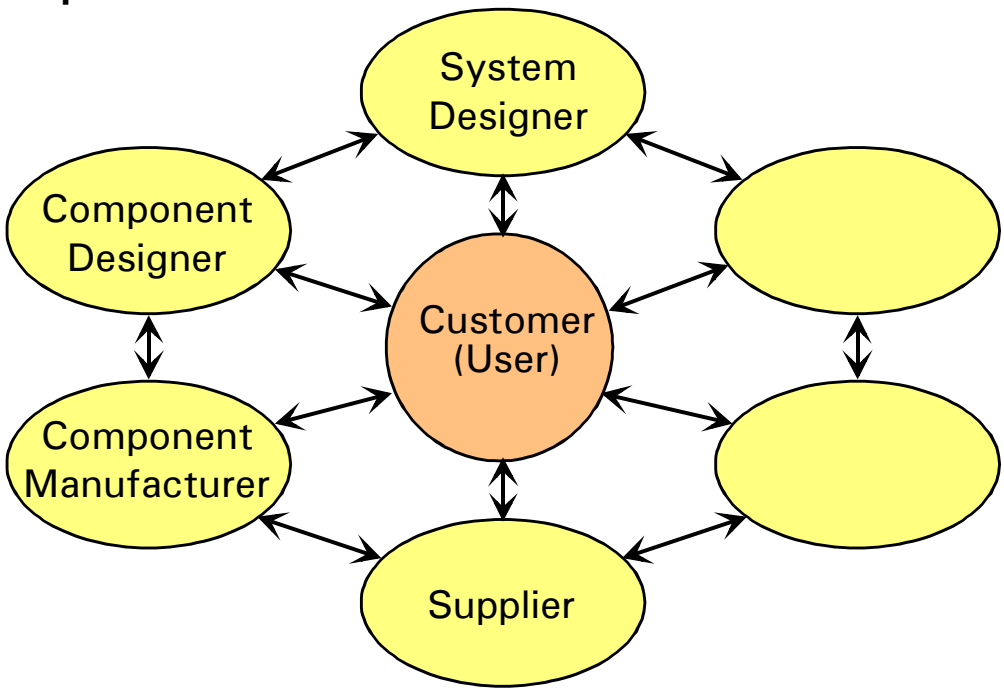
2. Designing a system means

- establishing **project goals**
- **defining the functions**, which the system shall perform
- and "**border conditions**" like size, cost, ... User's view

- -

- **planing** the development project Developer's view
- **checking and refining the requirements**
- **partitioning** the system (system concept/architecture), i.e.
  - identify, which components are needed and
  - assign functions to components
  - defining the internal interfaces between the components
- **developing** or purchasing the components

Involved parties



3. Development project types

Technical products can be divided in 3 major categories. Depending on the category, customer involvement during the project and intensity of the project phases differ very much:

- **Multiple customers - high volume products** ("mass product")

  e.g. TV sets, mobile phones, office standard software

– At the end of the development process the **product "waits" for customers**. The customer is not involved in the development process, a marketing department within the company defines technical features and target price.

– Analysis of **competitor products** is crucial.

– New products are **feature enhanced** old **products**.

– **Price**, **public relation/marketing** and **time to market** are often more **important** than technical features.

- **Single customer - high volume products**

  e.g. automotive electronics

– Product **design** is mostly **customer specific**. The **customer** is **heavily involved** in planning, requirements analysis, design decisions and testing. → Development partnership

– In many cases the product is an add-in to the customer's own product. In these cases the **development** process must be **synchronized to the customer**'s own development.

– **Further development** activities **during operation/maintenance** phase to prolong production lifetime and for continued parts cost reduction.

- **Single customer - low volume product**

  e.g. steel rolling mill, automobile production equipment

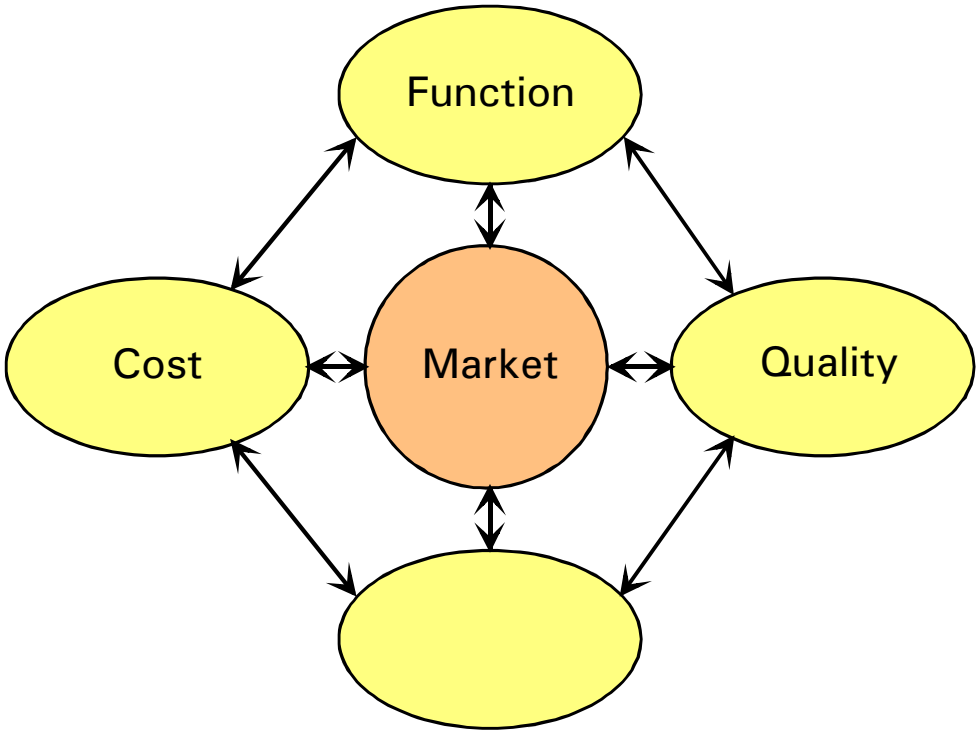
– Design is completely customer specific. → Engin. service

– Support during operation and maintenance of the "product" are as important as the "product" development.

A comparison:

	Multiple customers high volume products	**Single customer high volume products**	**Single customer low volume product**
Customer involvement	none	very important + + +	
Technical features	less important	very important	
Parts cost	very important + + +		not important – – –
Development time	short months	medium ... long 1 ... 5 years	long 2 ... 5 years
Development cost	low	very high	very important + + +
Time to market	very important + + +	Important	not important
Operation/ maintenance	no	product care cost reduction	very important
Product marketing	very important	not important	
Production lifetime	Short months	Medium 2 ... 5 years	Very long (incl. maintenance) 5 ... 10 years

4. Development project goals must be defined in terms of



Goals must be

e.g. PC development project

- **precise**
  – low cost PC
  + total cost below $500
- **measurable**
  – optimum performance
  + performance under defined test conditions, e.g. benchmark test
- **achievable**
  – gain 100% market share
  + increase our sales volume by 25%
- **consistent**
  – use newest Intel CPU
  + use CPU which costs < $200

Goals may be conflicting, especially between involved parties.

Example: Study MSc ITAE in Esslingen

Your goals:
1. Graduate with best grade in all subjects
2. Graduate in the shortest possible time
3. Earn $ 3000,- per month with student jobs

Will you be successful?

Most likely not!

Discussing the goals:

↓ Criteria Alternatives →

	Goal 1	Goal 2	Goal 3
precise?	Yes	No → specify time or date	Yes
measurable?	Yes	No → time or date: yes	Yes
achievable?	Only if you are a genius !?	Yes	No, students may only work <40hrs/month
consistent?		No, need more time to work	

Dealing with conflicting goals: **Assign Priorities**

e.g.

A (highest priority) **must** be a achieved
→ a 'killer' feature

B **shall** be achieved

C **may** be achieved
→ a 'nice to have' feature

How to set priorities?

- **Eisenhower's principle**

Don't confuse: **important** – **urgent**

	very important	less important
very urgent	A • start at once • need to do it yourself	B • look for help • let somebody do for you (delegate)
less urgent	C • keep for later • schedule in your activities list	W • forget about it ! • put into the waste paper basket

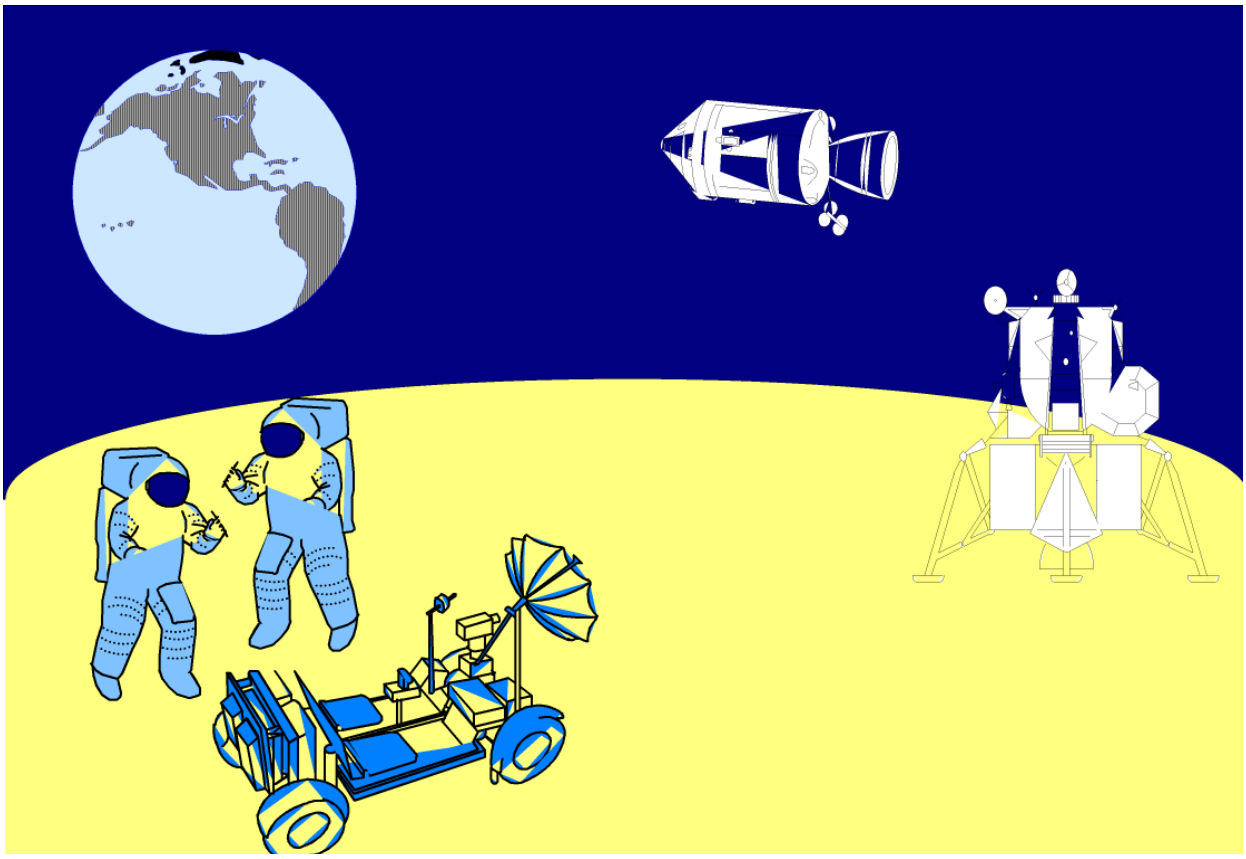
- **Pareto's rule: The 80% – 20% principle**

80% of the total result can be achieved with 20% of the total effort. The remaining 20% of the functions take more than 80% of the effort!

- Identify the 'must' functions → A
- The 'rest' goes into the B and C category
- Use incremental development:
  - Implement the must functions and create a working prototype with limited functionality
  - Add functions and features incrementally according to B and C categories
  - Finally optimize

B. Decision Making and Planning

1. The NASA game



Imagine the following situation:

- You are in a team of 2 astronauts, who have landed on the **bright side of the moon**.
- You are on an excursion with your lunar vehicle some **100 km away from your lander module** (the rocket, which will bring you back to your mother ship in the moon orbit).
- Unfortunately the lunar vehicle has broken down and you have to **walk back to your lander module**.
- For such cases you have an **emergency equipment**, consisting of the **12 items**.
- However, you cannot carry all these items with you and most likely during your march you will have to leave back items from time to time. So you have to **decide**, **which items are more important** for your survival than others.

The items in your emergency kit are (unordered list) :

- 90-110 MHz sender/receiver
- Parachute
- Matches
- First aid kit
- Stellar map (moon and stars)
- 20 m rope
- Signal pistol with flares
- Portable heater
- 20 l water
- Compass
- Two 5o kg oxygen tanks
- Food concentrate

These items shall be assigned **unique priorities from 1** (most important) **to 12** (least important). No two items must have the same priority.

The assignment is done in two rounds:

a) Each student delivers his/her individual solution.

b) Students discuss and a agree on a team solution with 4...5 students being in a team.

a) Individual solution:

Student name: _____

Items in my emergency kit:

Item	*priority *1*
90-110 MHz sender/receiver	
Parachute	
Matches	
First aid kit	
Stellar map (moon and stars)	
20 m rope	
Signal pistol with flares	
Portable heater	
20 l water	
Compass	
Two 5o kg oxygen tanks	
Food concentrate	

*1

Please **assign priorities from 1** (most important) **to 12** (least important). No two items must have the same priority.

b) Team solution:

Team members: _____

_____

_____

_____

Items in our emergency kit:

Item	*priority *1*
90-110 MHz sender/receiver	
Parachute	
Matches	
First aid kit	
Stellar map (moon and stars)	
20 m rope	
Signal pistol with flares	
Portable heater	
20 l water	
Compass	
Two 5o kg oxygen tanks	
Food concentrate	

*1

Please **assign priorities from 1** (most important) **to 12** (least important). No two items must have the same priority.

Observations:

- Decisions can not always be made based on pure facts and real knowledge → deal with uncertainty
- Decisions in groups can be influenced by communication skills of group members → deal with the "human factor"

2. A formalized, semi-objective way of decision making: Weighted criteria assessment

- **Define** the **alternatives**
- **Formulate** the **criteria** to assess the alternatives
  Use positive criteria rather than negative ones (if possible)
- **Assess** the **importance of** the **criteria** ("priority scale")
  Use a 'weight' scale, e.g. 10 for 'most important', 1 for 'least important'
- **Assess** the **alternatives** criterion by criterion
  Use a point scale, e.g. 10 for 'best', 0 for 'worst'
- **Multiply** your weights with your points **and sum up** for each **alternative**

An example situation:

- You are a FHTE student, looking forward to the 2 months summer break.
- Next semester you will start working on your master thesis.
- You have an old car in very bad condition. A new car would cost $1500 minimum.
- You have a friend, living in Madrid/Spain. Visiting your friend would cost $800 minimum.
- You have $1200.

What to do: Work and buy a new car or visit your friend?

Hints for using this method:

- Try to spread the weight and the assessment scale to better differentiate as much as possible.
- If sums are close to each other, the decision still is unclear. Review your weights or try to find more criteria.
- Remove those criteria, which nearly have the same meaning (e.g. “gain working experience” and “gain experience in foreign countries”) and try to reformulate one common criterion, including both goals (e.g. “gain experience which is helpful for your future job”).

  → By having to many criteria with nearly identical meaning the decision process can be easily manipulated in favor of one alternative.

- Try to find compromise alternatives.
- Don't trust blindly! Critically review your results. In case of unexpected results, check your criteria and weights.

It is the discussion process, forcing you to think about, express and understand, what you and the others in the team really want or feel, which makes this method valid, not the naked result values.

			Alternative 1 Work and buy a new car		Alternative 2 Visit your friend in Madrid		Alternative 3	
#	Criterion	Weight	Points	Points x Weight	Points	Points x Weight	Points	Points x Weight
	Ensure mobility	6	10	60	0	0		
	Recreate from student life stress	1	1	1	10	10		
	Relationship to your friend	4	0	0	10	40		
	Gain experience which is helpful in your future job	10	8	80	5	50		
			Sum:	141	Sum:	100	Sum:	

			Alternative 1		Alternative 2		Alternative 3	
#	Criterion	Weight	Points	Points x Weight	Points	Points x Weight	Points	Points x Weight
			Sum:		**Sum:**		**Sum:**	

3. Planning

... is the key to successful projects. Nearly all every day life activities include some amount of planning.

Examples:

- traveling:
  - Where to go?
  - Which routes to take to get there?
  - When does the bus/train/plane go?
  - Hotel/camp ground reservation made?
  - Passport valid, visa needed?
  - Credit card, foreign currency, traveler cheques available?
  - How many and which clothes needed?
  - . . .
- buying food:
  - What do you want to eat/drink?
  - Contents of refrigerator checked?
  - In which shops to buy?
  - "Transportation" capacity sufficient?
  - Enough money available?
  - . . .
- . . .

Planning consists of 3 tasks:

a) Collecting the data basis for planning

b) Scheduling activities (time table = "making the plan")

c) Continuos controlling and correcting the plan ("feedback")

3.1 Planning - step by step

a) Collecting the data basis for planning

1. Define **activities** - **what must be done ?**

   - Write down all activities, which must be performed during the project → **activity list**
   - Specify the expected result of each activity. Define, how to check, whether an activity was carried out successful.
   - Start with a brainstorming approach, i.e. write down activities without any order as they come into your mind.
   - Then try to group the activities, e.g. by assigning them to the different project phases (see chap. C).

2. Define **responsibilities** - **who does it ?**

   - For each activity write down, who has to perform it ("human resources"), i.e. who is responsible, that the desired results are available in time. Always assign humans ("personal responsibility") not organizations like companies, departments, ... !

3. Define **prerequisites** (interdependencies) - **what is needed before the activity can begin ?**

   - For each activity think about, which results from other activities will be needed, before this activity can begin (e.g. PCB design does only make sense, when the circuit design is completed and circuit diagram and parts list are available).

4. Define technical **resources** - **what is needed during the activity ?**

   - E.g. EMC test cell for EMC tests, prototype vehicle, ...

5. Define **needed/available time** - **how long does the activity take ?**

- For each activity estimate, how long it will take to carry out the activity.
- In some cases the available time for an activity (or the whole project) is forced by conditions outside of your control (e.g. road winter testing for vehicles in Europe is limited to December to March, your master thesis must be finished within 6 months, ...). This will define the amount of resources needed, to carry out all activities in the available time or limit the number of activities, which can be carried out.

b) **Scheduling the activities**

6. Define **the sequence** **- when will it be done ?**

- The interdependencies of the activities (step 3) and the human (step 2) and technical resources (step 4) define a logical sequence of the activities.
- Activities, which do not conflict with respect to resources and interdependencies can be paralleled to reduce the overall time.
- Together with the needed or available time (step 5) for each activity a → **time schedule** (the "plan") can be established. Time schedules typically are presented as GANTT or PERT diagram.

c) **Continuos controlling and correcting the plan ("Feedback")**

- The begin and end of all activities must be monitored and the results checked. If the reality and the plan differ (the usual case), the plan must be corrected.
- At the beginning of a project, the activities may be rather coarse, summarizing rather than going into detail. During the project details will be worked out step by step. Typically at the end of each phase, the next phase(s) are planned in more detail.

3.2 A planning example

Establish a coarse plan for the development of a mobile phone.

Note:
To simplify the plan, activities for
- market analysis,
- marketing campaign and introduction
- production preparation
- documentation and certification
- . . .

will not be considered here.

Remarks:

- Activity lists and schedules are partially redundant. Activity lists are better suited for details. Schedules are better suited to show the sequence of the activities.
- Often various levels of detail are necessary, e.g. for management a schedule with major phases is sufficient, whereas for engineering staff activities must be broken down to details.
- To reduce redundancy, allow various views on the data and easily adapt the plan to any changes during the project, **project management software** like MS Project is available. Other tools like Outlook or PDA software may also offer partial planning support.

However: Planning is still an 'engineering' task and cannot be done automatically!

Activity list

#	What?	Who?	How long?	When?		Done?
	Activity - Prerequisites (PR) - resources (RS) - results (R)	Responsibility	Duration	Begin	End	Check
1	Requirements analysis PR: Marketing analysis, R: Req.Doc	Marketing Prod.Planning	2w	See Schedule		
2	Concept phase PR: 1, R: System Architecture and Spec. Doc	HW, SW, ME concept team	2w			
3	Mechanical design PR: 2, R: Case prototype	ME dept.	3w			
4	Hardware design PR:2 + (3), R: PCB prototype	HW dept.	6w			
5	Mechanical + HW integration PR: 3 + 4, R: Working prototype HW	HW, ME	1w			
6	Software design PR: 2, RS: Breadboard HW, R: SW prototype (?)	SW dept.	8w			
7	Hardware + software integration PR: 4 + 6, Working prototype	HW, SW	2w			
8	Prototype test PR: 5 + 7, R: Test documentation	Test Team	4w			
9	Redesign/Retest (details like 4...9) PR: 9, R: Production ready product	HW, SW, ME	4w			

Schedule

#	What?	Who?	When? (scale is CW=calender weeks)																									
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	Req. analysis	MG, PP																										
2	Concept phase	Concept T.																										
2.1	Concept approval	Chief Eng.																										
	Implementation																											
3	Mechanical design	ME																										
4	Hardware design	HW																										
5	ME/HW integration	HW, ME																										
6	Software design	SW																										
7	HW/SW integration	HW, SW																										
7.1	Prototype release	Chief Eng.																										
8	Prototype test	Test T.																										
8.1	Test evaluation	Chief Eng.																										
9	Redesign/Retest	All																										
9.1	Release for Prod.	Chief Eng.																										

Legend: △ Begin of activity ▽ End of activity ◇ Milestone ▲▼◆ Done

Activity list

#	What?	Who?	How long?	When?		Done?
	Activity - Prerequisites (PR) - resources (RS) - results (R)	Responsibility	Duration	Begin	End	Check

Schedule

#	What?	Who?	When? (scale is)																								
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Legend: △ Begin of activity ▽ End of activity ◇ Milestone ▲ ▼ ◆ Done

3.3 Keys to successful planning

1. Agreement with all involved parties

- If others are involved in your project to carry out activities, deliver input for your project or use your results, let them specify their own activities and the time needed for them.
- No project will succeed, if not all involved parties agree to their part of the activity list and time schedule.

2. Don't overconstraint the plan

- A plan is a **tradeoff between activities, resources and time**. Not all of these 3 items may be predefined:

- If activities and resources are predefined, the time is an output of the planning process rather than an input.
- If resources and time are predefined, the possible activities are an output, resulting in limited or canceled activities or activities carried out with limited quality.

3. Don't use best case planning

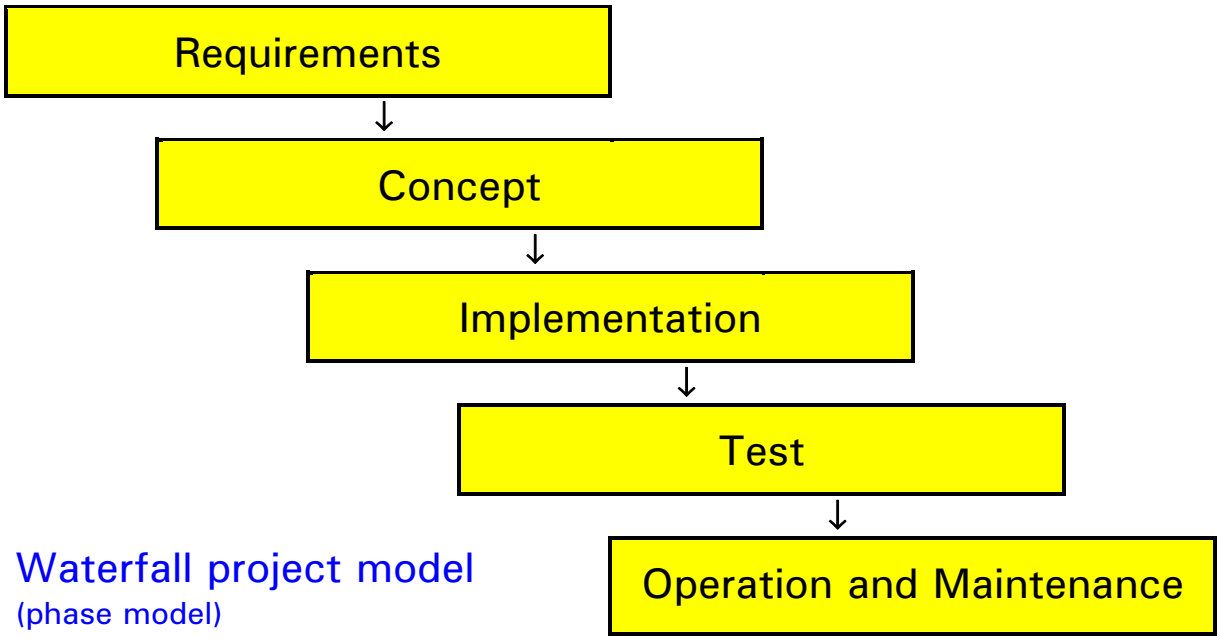
- Leave **room for the unexpected** in your plans! If anything can go wrong, it will ... (Murphy's law). "First time right" and "zero failure" are good goals, but hard to achieve.
- Include **milestones**, where to **check results** and include phases for correcting them ('**redesign**'). The time between the milestones shall be as short as possible (use 'sub-milestones'), to make the feedback loops small.
- **Sequence** activities **according to** their **priority**, put less important activities at the end! If the project runs out of time, cut or shorten these activities.

4. Build up planning experience

- Always review your projects to learn for the next project. What was good, what went wrong? Which was the actual time needed? Which partners were reliable?

C. Requirements Analysis and Project Phases

1. A simple project model



Waterfall project model
(phase model)

Each phase has

- **activities**

... to be performed to produce the phase's result
e.g. in the implementation phase:
circuit development, PCB development, ...

- **results**

- "products", documenting the results of the phase
= input for next phase
e.g. at the end of the requirements phase:
hardware/software specification document
e.g. at the end of the implementation phase:
fully functional prototype

- **milestone(s)** or gate(s)
  - result(s) which must be achieved and (formally) be approved before proceeding to the next phase
  e.g. at the end of the test phase: release for production

In big projects each phase will be subdivided in subphases with parallel and sequential activities.

2. Requirements analysis

2.1 Example: Walkman AC Adapter

Original requirements specified by the customer:

- Develop an AC adapter for a walkman, to connect the walkman to the mains power supply. The adapter will be sold in Europe.

Refined requirements

- After the designer has interviewed the customer, a detailed requirements document can be written:

Requirements Document for FHTE-Walkman AC Adapter

1. Purpose

This paper describes an AC adapter for our FHTE-walkman, to connect the FHTE-walkman to the mains power supply.

2. Marketing Information

The product is to be sold as an option to those customers, who also buy our FHTE-walkman. Competition and their products' features were already described in the FHTE-walkman marketing study. However, the AC adapter will only be sold in Europe.

3. Function

3.1 Normal Operation

The AC adapter shall deliver a constant DC output voltage at a variable output current within the specified AC input voltage range. There shall be no user selectable or adjustable parameters.

3.2 Startup and Shutdown Procedure

The AC adapter shall start and stop operation, whenever the AC input connector is plugged into or removed from a mains socket. The load may or may not be connected during startup

and shutdown and may be connected or removed during normal operation.

3.3 Failure Behavior

The adapter's output shall be short circuit protected. The adapter may turn off its output voltage, if the adapter's internal temperature becomes too high. It must turn on again after cooling down, if the input plug is removed and reconnected.

3.4 Additional Functions

None

4. Technical Data

4.1 Electrical and Mechanical (see also 3.2) Interface

• Input	110 ... 230V ± 10% / AC 50 Hz 2pin Euro plug with 2.5m cable
• Output	9V DC ± 3%, 0 ... 500mA isolated from input 3.5mm socket

4.2 Case

• Material	non-conductive plastic
• Color	black or gray
• Insulation:	double insulated, comply with VDE 100
• Size	< 6 x 6 x 7 cm³
• Weight	< 250g
• Surface temperature	< 50°C

4.3 User Interface

• Switches and keys	none, turn on by plugging in the input cable
• Display, control lamps	LED (green), if output voltage is in specified range, brightness t.b.d.

4.4 Environmental Conditions

- Ambient temperature: +10...30°C
- Air humidity: 10 ... 80%
- Sealing: IP 44
- Electromagnetic compatibility: CE
- Vibration and shock: 1g sine wave 1 ... 1000Hz
  must withstand a 1m fall
  on concrete floor

5. Commercial Data

- Projected sales volume: 10000 (1st 3 months)
  100 000 (total in 3 years)
- Projected parts cost: < $ 20
- Projected availability: Preproduction samples May 2004
- Start of Production Sep. 2004

2.2 Requirements for 'Requirement Documents'

- Requirement documents should be **written from a user**/customer's **view**, **defining, what the system**/product **should do**, not how it is implemented.

  During the **concept phase**, the requirements document is **converted into a 'Specification Document'**, **which describes, how the system**/product **shall be implemented** together with a detailed project plan and a parts and development cost estimation.

Required contents:

- **Functions** of the system **in all operating modes.** Many customers forget to specify startup and shutdown procedures and the desired behavior in failure situations
- **Electrical and mechanical interfaces description**
- **User interface** description (how to operate the system)
- **Environmental conditions** (ambient temperature, sealing, vibration, EMC, ... for hardware; computer platform, operating system, disk/ram sizes, ... for software)
- A requirements document **should refer to or include** a short abstract of **a marketing study**, showing the main competitors, market share and competing products with key features and prices. Very often such a marketing study initializes the product development project.

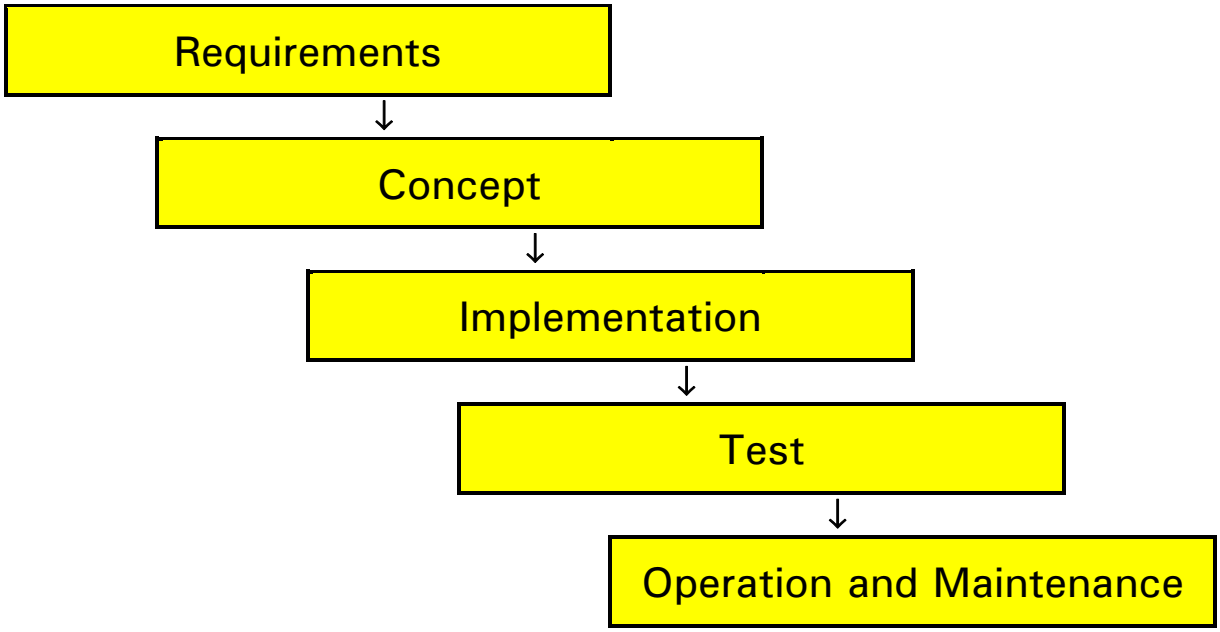
Style

- **Use data** rather than purely describing properties (minimum and maximum values or nominal values and tolerances if possible)
- **Use established standards** (e.g. IEC, ISO, ANSI, ...), where applicable
- **Include all items necessary for product development**. If these items cannot yet be defined, mark them as "t.b.d = to be defined later".

Intentionally left blank

3. Real world projects

3.1 Problems with the waterfall project model



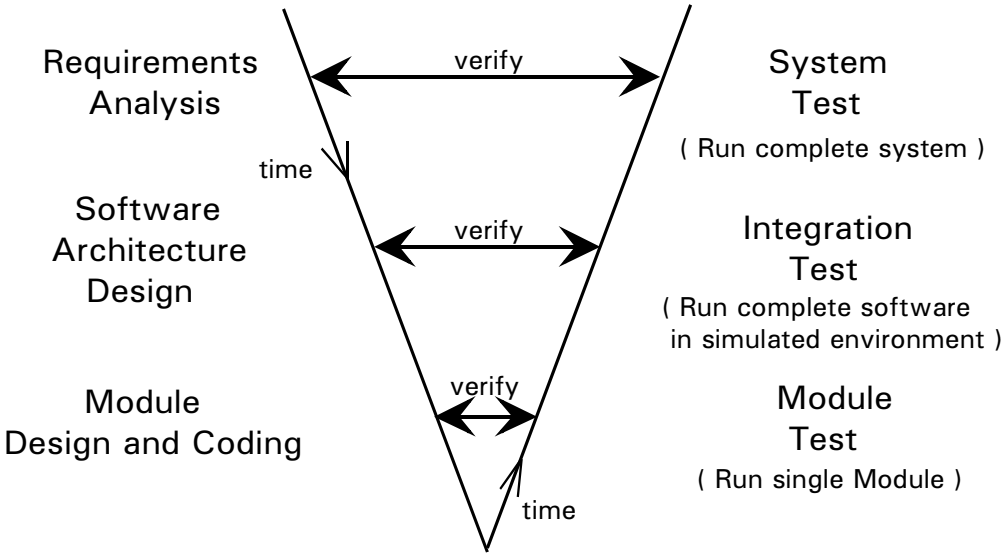
The **waterfall model** is strictly **sequential** ...

Nr.	Aufgabe	Aug	Sep	Okt	Nov	Dez	Jan	Feb	Mär	Apr	Mai	Jun	Jul	Aug
							2000							
1	Requirements analysis													
2	Project Decision													
3	Concept phase													
4	Concept approvement													
5	Implementation phase													
6	Prototype release													
7	Test													
8	Production release													
9	Operation and Maintenance													

. . . **real world** projects are not ! In real world projects

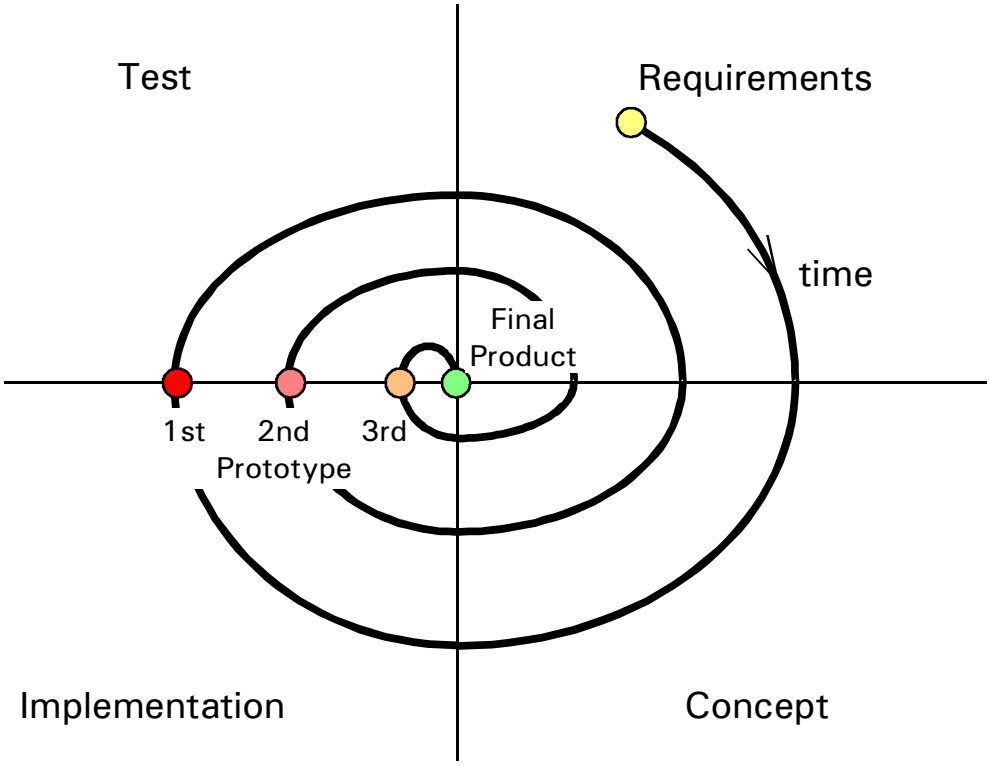
- due to missing information **not all details** for later phases **can be specified in early phases**
- due to technical problems **implementation tradeoffs have retrospective effects** on the concept or even on the requirements
- **time or resource constraints do not allow** a fully **sequential schedule**
- **customer or market demands** may **change** (the longer the project, the more this is likely)

- Long feedback loops V-model of software dev. BAD



Errors in the requirements analysis (first phase of the development process), causing a wrong system behavior, are often only found during the system test (at the end of the development process)

Spiral Model: Incremental design instead of "first time right"



- Incremental design: Plan to refine requirements and prototype function in a carefully planned step-by-step process

3.2 Improving the development process

- **Top-Down vs. Bottom-Up**

Top-down: Proceed from a global overview (goals, objects, block diagram level, interfaces,...) to the detailed level (algorithms, circuits, ...)

Bottom-up: Proceed from the detailed implementation level to the complete system

Example: Items to solve when designing a telephone network

TOP	How many users at which locations? How often and how long will the calls between the different locations be? ...
MIDDLE	Wireless or wire-bound transmission? Optical or electrical? Analog or digital? ...
	Class A or class A/B audio amplifier?
BOTTOM	RJ45 or TAE connector?

– Use a **top-down** approach **to get the big picture** fast (requirements, concept, major building blocks)

– **Identify the critical** technical **problems** ("job killers")

– Use a **bottom-up** approach **to find out** early, **if and how the critical technical problems can be solved**

→ make a good concept and eliminate risks early

- **Rapid prototyping simulation and implementation tools**

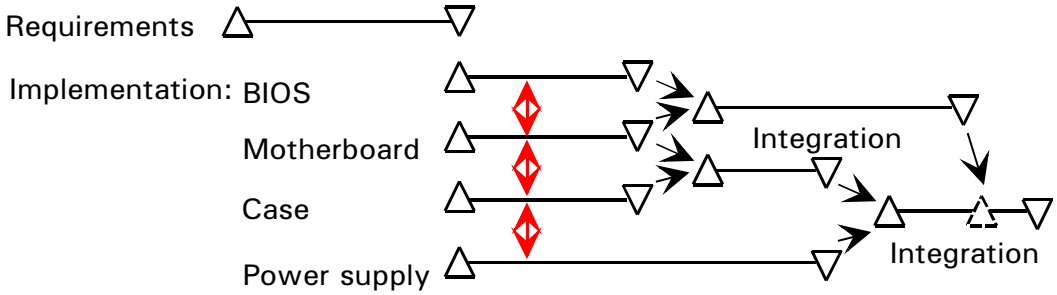
– Requirements and concept are verified by simulation rather than waiting until the implementation is completed.

– Prototypes are build by automatic generation tools (e.g. program code synthesis from UML models, circuit synthesis from VHDL/Verilog descriptions)

→ reduce errors in requirements and implementation

- **Simultaneous engineering (Parallel design)**

– Split up the development between simultaneous working groups, e.g. PC development → HW/SW co-design



→ reduces overall development time

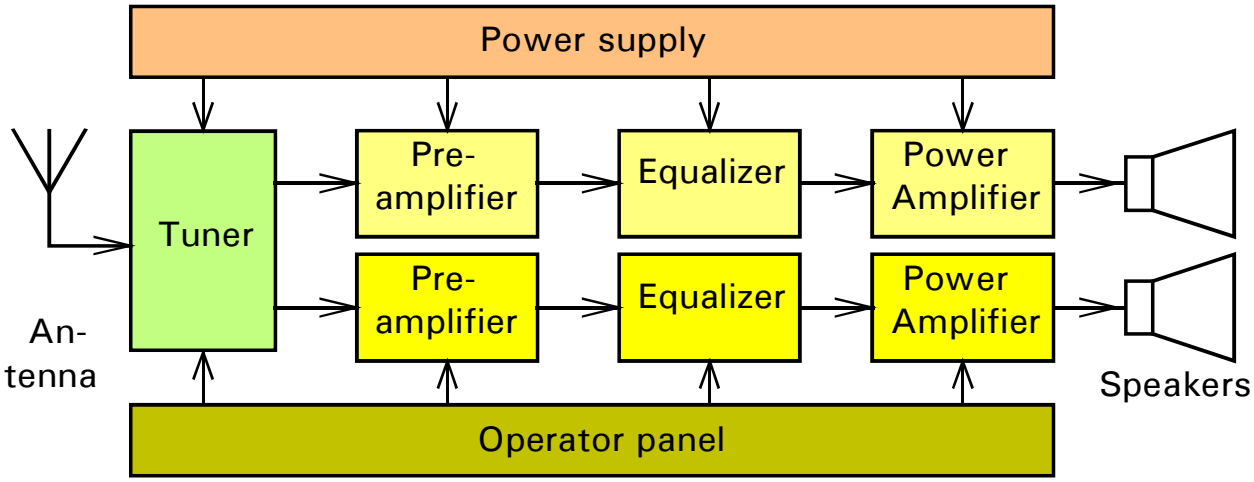
– Simultaneous engineering requires:
  * well-defined interfaces between the modules
  * well-defined information flow between design groups

4. System Architecture, Partitioning and Interfaces

Examples

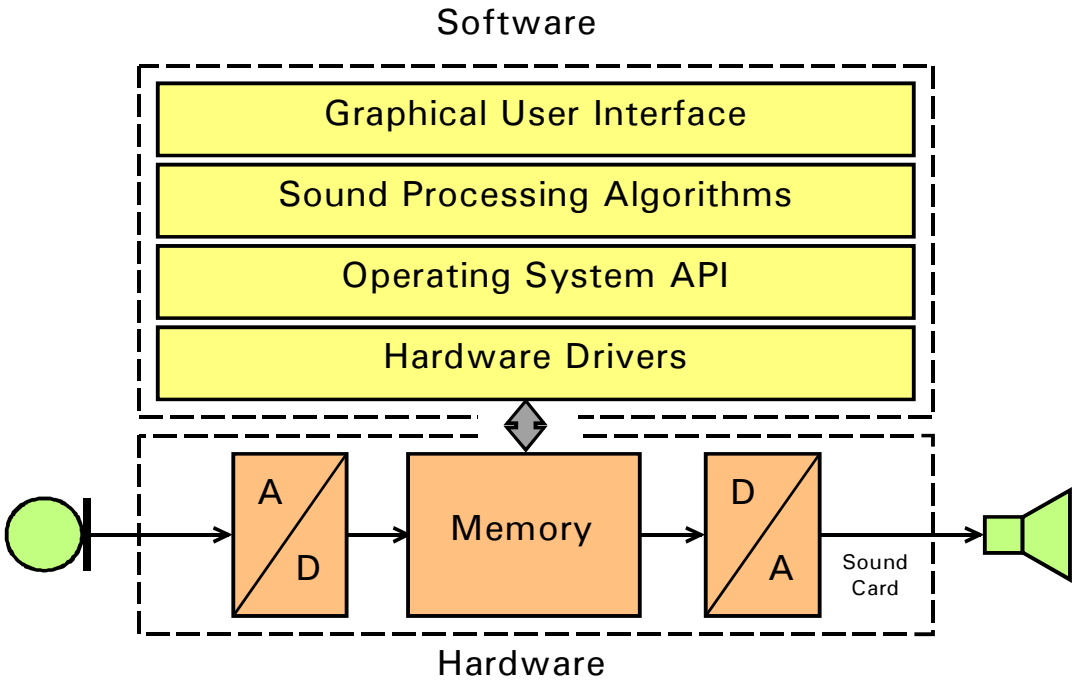
Horizontally and vertically structuring a system in blocks (modules) and layers.

- Stereo radio set



Horizontally structured (block diagram structure)

- PC based sound processing



Vertically structured ("layered")

Partitioning / structuring principles

- Signal flow centered partitioning
  - combine functions which lie in the same signal path (signal flow centered) e.g. left channel – right channel
- Function oriented partitioning:
  - Combine similar functions
    e.g. preamplifier – equalizer - ...
    e.g. physical layer – core logic – user interface

Design rules for partitioning and interfaces:

- Modules shall have clearly defined functions and interfaces
- Modules shall be independent (during development, manufacturing, test, maintenance, ...)
- Separate "stable" and "variable" functions
- Realize in software, what needs to be field upgradable
- Interfaces shall be "small and simple" (low number of signals, parameters, ...)
- Interfaces shall be robust (hardware: short circuit / overvoltage protected, software: parameter checking, error codes, ...)
- Don't reinvent the wheel, use established standards for interfaces and standard hardware/software modules (where applicable, especially in low volume products)!

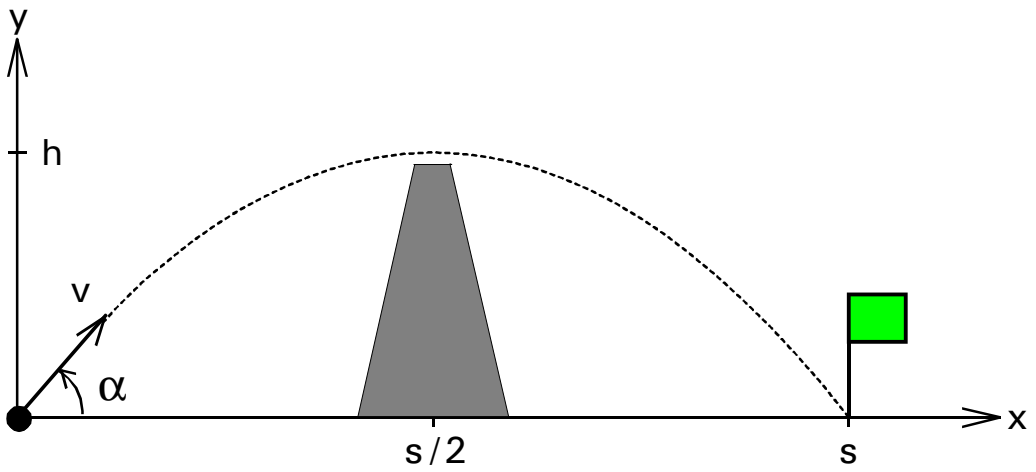
Design and partitioning rules apply equally to hardware and software → see Software engineering module in Information Technology.

Module Neural Networks

Neural Networks – Application in Control Engineering

1. Application Example

- A golf ball shall be driven over a distance s to meet the target green. Unfortunately, there is a row of bushes with height h in the middle between the golf player and the target green. Which driving angle α and which driving speed v is necessary, to directly hit the target?

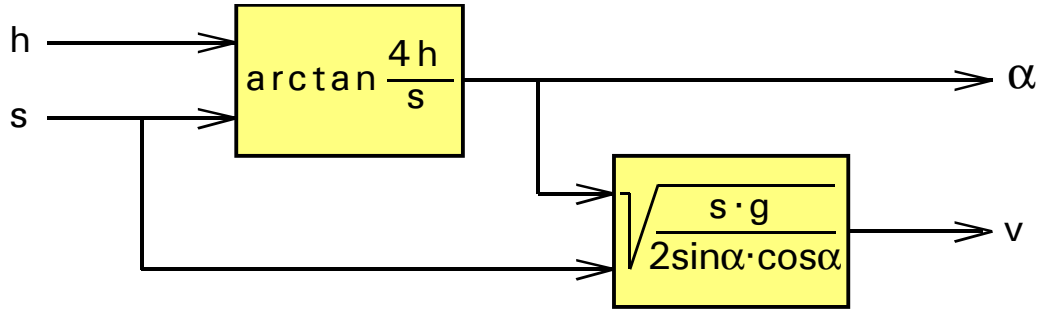


Solution using a classical open loop control algorithm:

- Under ideal conditions, i.e. neglecting the air resistance and the ball's spin, the ball's flight curve can be described by a parabolic trajectory:

$$x(t) = v \cdot \cos\alpha \cdot t \qquad (1) \qquad\qquad y(t) = v \cdot \sin\alpha \cdot t - \frac{1}{2} \cdot g \cdot t^2 \qquad (2)$$

For a given set of s and h, the necessary speed v and angle α can be computed from equations (1) and (2) and converted into an appropriate **open loop control algorithm**:



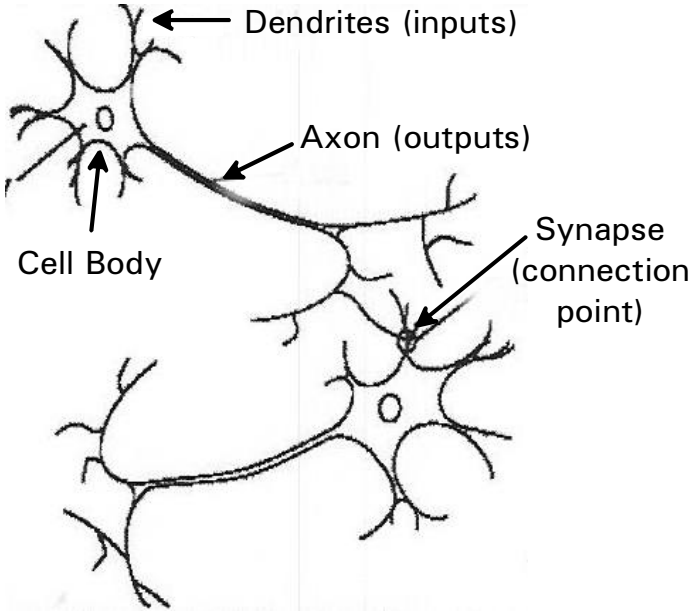
Nevertheless, this solution is problematic:
Due to the air resistance and the spin of the ball, the ball's real trajectory will not be an ideal parabolic curve. So with an open loop control algorithm some **empirical correction factors** will be necessary.

How does a human being solve such a problem?

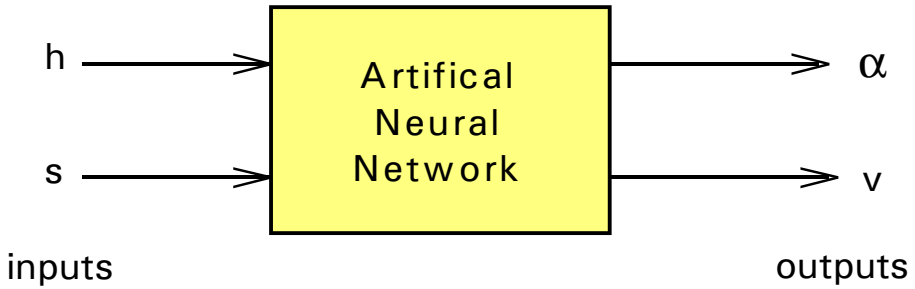
- Do some **training**:
  - Drive golf balls with various different values of v and α
  - **Learn** which height h and distance s the trajectory has and **correct** v and α, until a given set of s and h can be achieved.
- **Practice** golf:
  - Based on the learned values play golf. Interpolate and extrapolate experience to different sets of s, h.

Neural Networks try to model the human being's approach using a simple mathematical model of the human brain:

Biological network

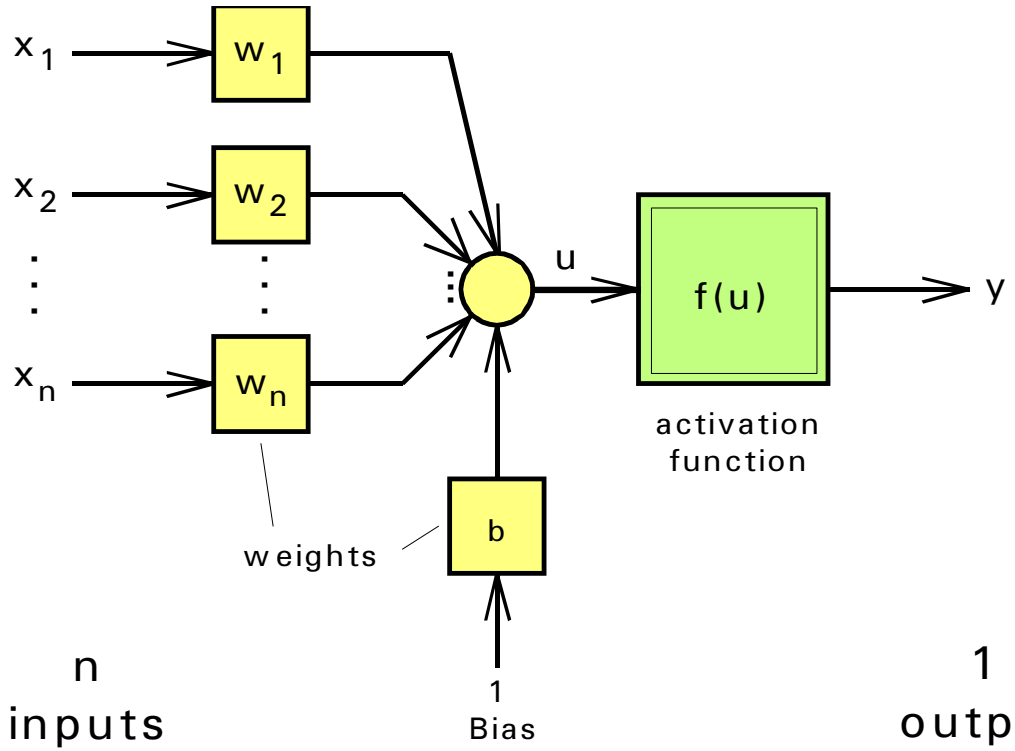


Artificial neural network



- Simple, slow processing basic element (**neuron**) approx. 1ms
- Many of these basic elements working in parallel (**parallelism**) approx. 10^{11}
- Many interconnections between the basic elements (**network**) approx. 10^{4} per neuron
- Nonlinear behavior of the elements with thresholds *Data of human brain*
- Learning by training does 'strengthen' or 'weaken' the connections
- Capability for generalization (interpolation and extrapolation) from trained experience

2. Artificial Neurons (Perceptron, Frank Rosenblatt 1957)



$$y = f(u) = f(\underline{w}^T \cdot \underline{x} + b)$$

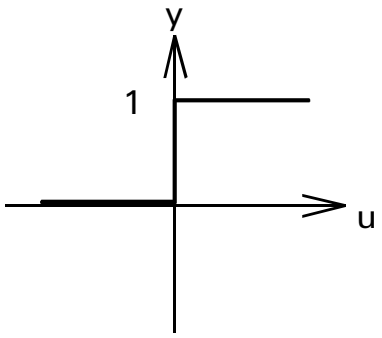
$\underline{x}$ ... column vector of n input signals

$\underline{w}^T$ ... row vector of n weight factors

The biological neuron is modeled as the weighted sum u of n input signals $x_1,...x_n$ plus a bias value b. The sum is converted into an output signal y by means of a (probably) nonlinear function f(u).

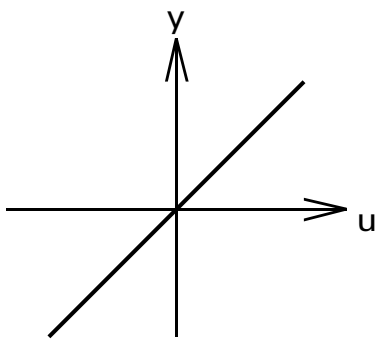
Please note: In neural network literature f(u) sometimes is named as 'transfer function', even if it does describe a static input-output relation. Don't mix this up with classical Laplace transfer functions which describe dynamic behavior.

Theoretically, f(u) may be an arbitrary function. For practical use in control engineering, f(u) should be a monotonous increasing, differentiable function. Commonly used functions are:



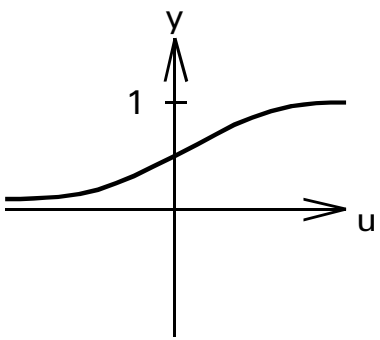
Hard-Limiter
(Matlab: hardlim())

not suited for control, used in classification



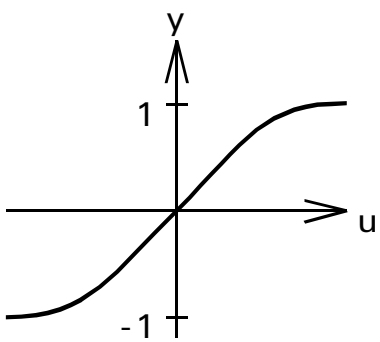
Pure Linear
(Matlab: purelin())

$y = u$



Log-Sigmoid
(Matlab: logsig())

$$y = \frac{1}{1 - e^{-u}}$$



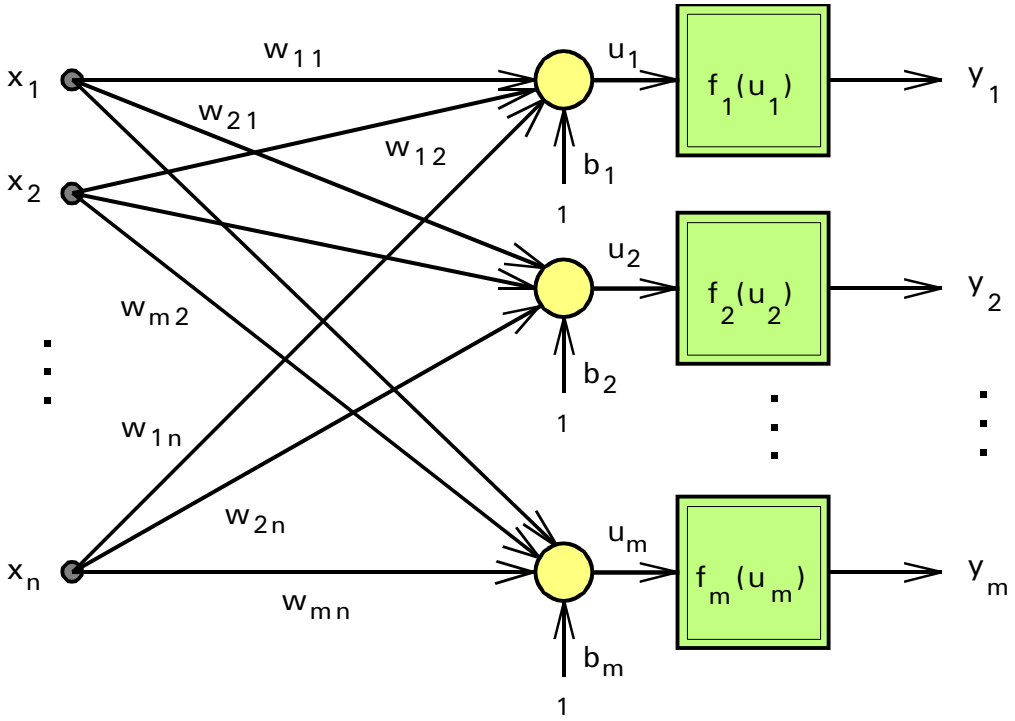
Tan-Sigmoid
(Matlab: tansig())

$$y = \frac{e^{+u} - e^{-u}}{e^{+u} + e^{-u}}$$

Theoretically, input signals of various magnitude can be adapted by appropriate weighing factors. However, to avoid numerical implementation problems, **input signals should be normalized** into a 0 ... 1 or −1 ... +1 range. When using normalized values, the absolute value of a weighing factor directly indicates the 'importance' of its associated signal.

3. Single Layer Feedforward Neural Network (Single Layer Perceptron)

To generate m output signals out of n input signals, a single layer neural network with m neurons can be used:



n inputs layer with m neurons m outputs

$$\underline{y} = \underline{f}(\underline{u}) = \underline{f}(\underline{W} \cdot \underline{x} + \underline{b})$$

$\underline{x}$ ... column vector of n input signals

$\underline{y}$ ... column vector of m output signals

$\underline{b}$ ... column vector of m bias values

$\underline{W}$ ... m x n matrix of weight factors

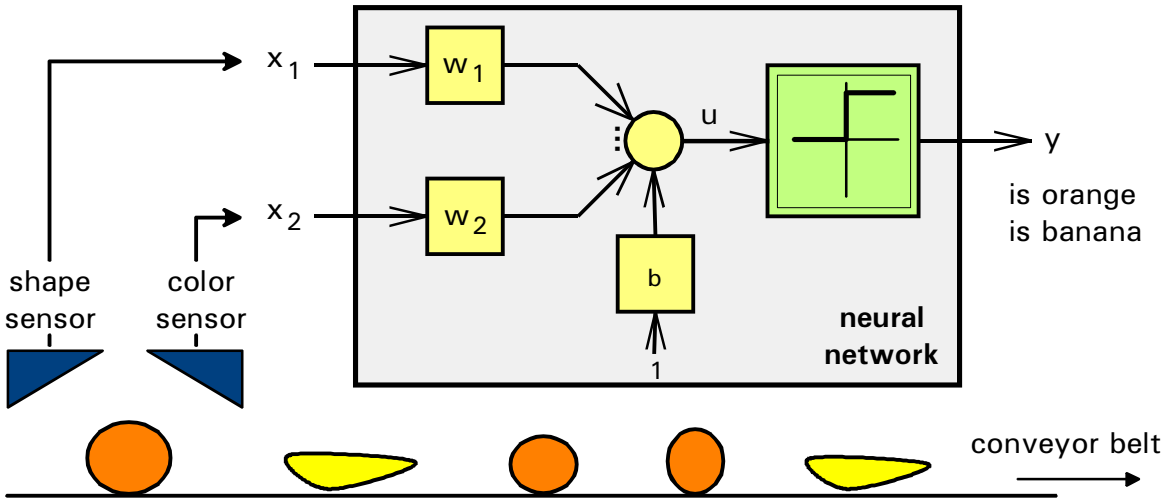
Typically, all m neurons use the same activation function f(u).

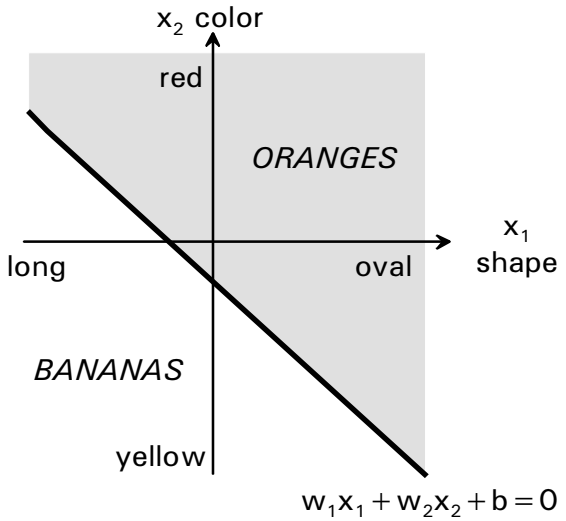
Please note: To keep the drawing simple, the input signal connections have been marked by the weight factors w_{ik} (i ... index of the output signal y_i, with k ... index of the input signal x_k), rather than drawing separate gain blocks.

Single layer feedforward nets with hard limiting activation functions (*Perceptron net*) **often** are **used for classification**.

Example: A single neuron is used to sort and classify fruit into the two classes 'orange' and 'banana'. Two properties of the fruit are measured and are used as input signals to the net:

x_1:	shape	with an oval shape giving a more positive signal than a long shape
x_2:	color	with red giving a more positive signal than yellow





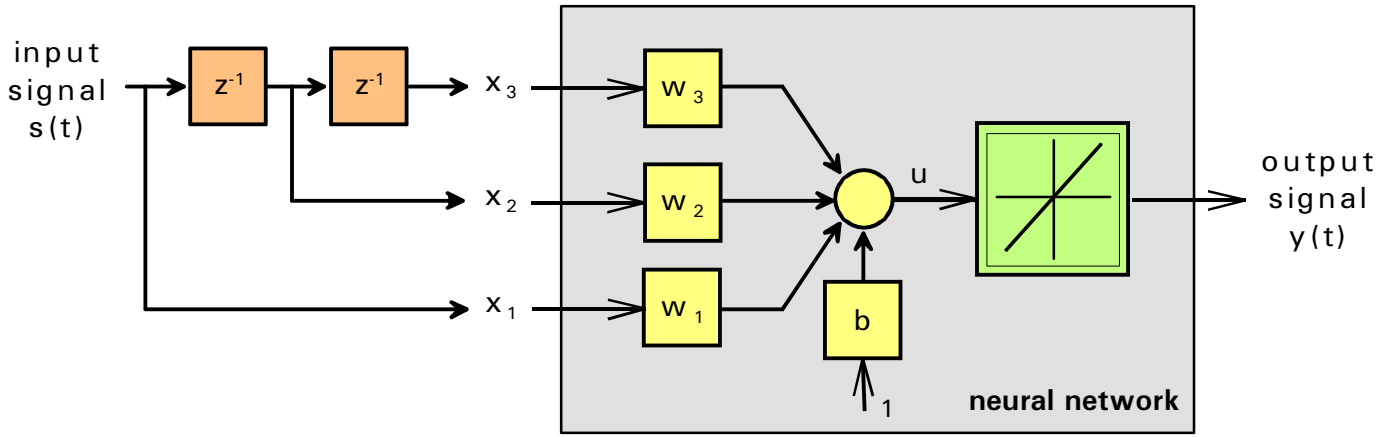
The hard limiting activation function splits the $x_1 - x_2$ plane into two parts with the border defined by w_1, w_2 and b.

With m neurons, classification with 2^m classes is possible.

The single layer feedforward net (no internal feedback paths) can generate arbitrary linearly separable functions (example of a function which is not linearly separable: XOR).

Single layer feedforward nets with linear activation functions (*Adalin net*) can be **used as signal processing filters**.

Example: The input signal is fed into a tapped chain of unit delays. The single neuron network is connected to the taps.



Using z-transform analysis, the networks output signal is

$$Y(z) = (w_1 + w_2 z^{-1} + w_3 z^{-2}) S(z) + b$$

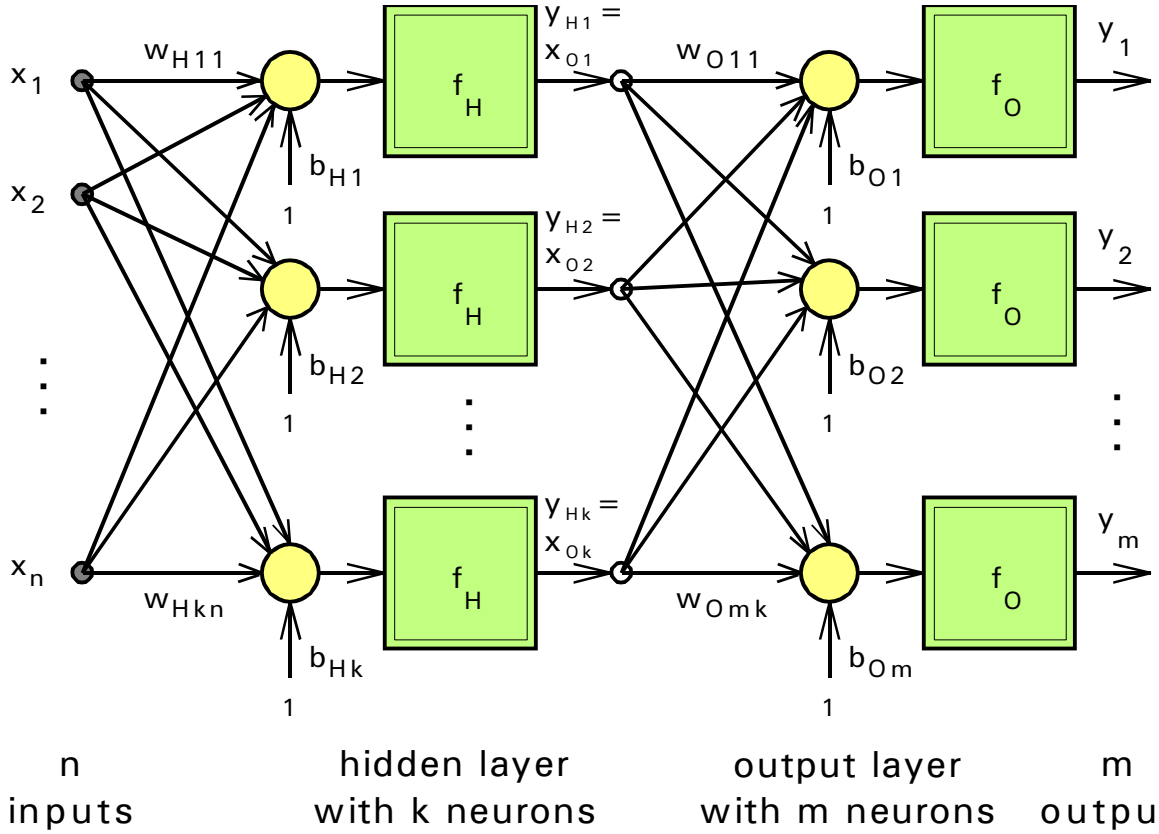
$$\rightarrow \quad \text{for } b = 0 \quad \rightarrow \quad G(z) = \frac{Y(z)}{S(z)} = w_1 + w_2 z^{-1} + w_3 z^{-2}$$

This is the well-known characteristic of a digital signal processing FIR filter.

By also feeding y(t) into a tapped delay chain and connecting the taps to net inputs, IIR filters can be implemented as well.

4. Multilayer Feedforward Neural Network (Two Layer Perceptron)

To overcome the limitations of the single layer network, two layers can be combined:



Hidden layer:

$$\underline{y}_H = \underline{x}_O = \underline{f}_H (\underline{W}_H \cdot \underline{x} + \underline{b}_H)$$

$\underline{x}$ ... column vector of n input signals
$\underline{y}_H$ ... column vector of k hidden signals
$\underline{b}_H$ ... column vector of k bias values
$\underline{W}_H$ ... k x n matrix of weight factors

Output layer:

$$\underline{y} = \underline{f}_O (\underline{W}_O \cdot \underline{x}_O + \underline{b}_O)$$

$\underline{y}$ ... column vector of m output signals
$\underline{b}_O$ ... column vector of m bias values
$\underline{W}_O$ ... m x k matrix of weight factors

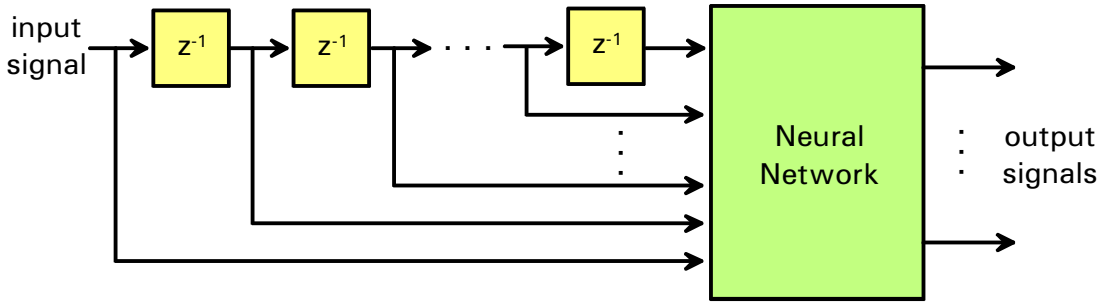
- The number of neurons in the hidden layer can be freely chosen, typ. $n \neq k \neq m$
- The hidden layer and the output layer often use different activation functions $f_H()$ (typically sigmoid) and $f_O()$ (typically linear)

Note:

- Many activities in the scientific community deal with neuronal networks. So **many different network structures have been proposed** (and will be in the future), e.g. multilayer feedforward networks or networks with internal feedback. Each of these structures does have its special advantages and disadvantages and is suited for a special application.
- **Only** the **single and multilayer feedforward networks** will be **discussed here**, because they are the oldest, best understood and most frequently used structures.

Implementing dynamic behavior in feedforward neural networks

- Multilayer feedforward networks can realize arbitrary nonlinear, however **static** functional **relations** between inputs and outputs. To implement dynamic behavior, the input signal(s) can be connected via a **chain of unit delays** (similar to an nonlinear FIR filter):



- Alternatively the network may use internal feedback, however, as is well-known from feedback control systems, this can cause stability problems.

5. Learning Process

The real charm of neural networks comes from the fact, that in contrast to state space controller e.g. the parameters need not be designed using complicated theoretical methods but can be "learned by example" in a training phase. Theoretically there a three types of learning:

- **Supervised learning**: A category of learning methods, where the **network** is **provided with** a **set of input values and expected (=target) output values** $\{\underline{x}^{(1)}, \underline{y}^{*(1)}\}$, $\{\underline{x}^{(2)}, \underline{y}^{*(2)}\}$, $\{\underline{x}^{(3)}, \underline{y}^{*(3)}\}$, . . . and the **weight and bias factors $\underline{W}$ and $\underline{b}$** are **tuned as to minimize the** (sum of the squared) **error between the expected output values $\underline{y}^*$ and the real output values $\underline{y}$**. This is the natural form of training for neural networks
- **Reinforcement learning**: A category of learning methods for situations, where no target output values are available, but some sort of grade value, which summarizes the "quality" of the output values in one scalar. During the training process the weight and bias factors are tuned in order to optimize this grade value. Such training methods are only applicable to certain kinds of problems and are therefore rarely used, but find some interest in control engineering, where e.g. the integrated absolute error (ITAE) can be used as a grade.
- **Unsupervised learning**: Theoretically, there is also the possibility, that the network learns from input values only without any target output values or grades, but to present these methods are investigated in research with no practical applications in control engineering.

Only supervised learning will be presented here. For details about the other types see the literature.

5.1 Learning Process for Single Layer Feedforward Networks: Delta Rule by Widrow-Hoff

Delta rule for supervised learning:

- Initialize the weight and bias factors with arbitrary values
- Apply a (randomly selected) set of input variables $\underline{x}$
- Let the network compute the output variables $\underline{y}$
- For the next iteration correct the weight and bias factors based on the error $\underline{e} = \underline{y}^* - \underline{y}$:

$$\underline{W}_{corrected} = \underline{W} + \Delta\underline{W} = \underline{W} + \eta \cdot \underline{\delta} \cdot \underline{x}^T \qquad \text{with } \underline{\delta} = \underline{\underline{F'(u)}} \cdot (\underline{y}^* - \underline{y})$$

$$\underline{b}_{corrected} = \underline{b} + \Delta\underline{b} = \underline{b} + \eta \cdot \underline{\delta} \cdot 1$$

$\underline{\underline{F'(u)}}$ is a m x m diagonal matrix with elements $\partial f(u)/\partial t|_{u=u(\underline{x})}$ on the main diagonal.

- Repeat the above procedure, until the error $|\underline{e}|$ for all available sets of input variables is small enough.

$0 < \eta << 1$ is the so called **learning rate**. Small values lead to slow convergence, big values may lead to instability. Frequently the learning rate is adapted during the learning process starting with a big value and gradually decreasing it.

It has been shown by Widrow and Hoff, that with a suitable small η this algorithm will always converge, provided that the given problem can be solved using a single layer feedforward network. **If f(u) can't be differentiated** (as with hard limiting activation functions), **use $\partial f(u)/\partial t = 1$.**

See the literature for Widows-Hoff's proof. For a mathematical explanation of the delta rule see the appendix.

Example: Orange-Banana Classification (continued from page A.7, see Fruit.m for a Matlab program)

Training data sets	*Shape: x_1* *(−1 = long, +1 = oval)*	*Color: x_2* *(−1 = yellow, +1 = red)*	*Category: Y* *(0 = banana , 1 = orange)*
Data Set A	−1	−1	0
Data Set B	−1	+1	1
Data Set C	+1	−1	1
Data Set D	+1	+1	1

Initial weights and bias: $w_1, w_2 = 1$, $b = -1$, Learning rate $\eta = 1$ (too big, but manual calc. here)

Step → Data Set	x_1	x_2	y^*		w_1	w_2	b	$u = w_1x_1 + w_2x_2 + b \rightarrow y = f(u)$	$\Delta w_1 = \eta\,(y^*-y)\,x_1$	$\Delta w_2 = \eta\,(y^*-y)\,x_2$	$\Delta b = \eta\,(y^*-y)$
1 → A	−1	−1	0		1	1	−1	−3 → 0	0	0	0
2 → B	−1	+1	1		1	1	−1	**−1 → 0**	**−1**	**+1**	**+1**
3 → B	−1	+1	1		**0**	**2**	**0**	**+2 → 1**	**0**	**0**	**0**
4 → C	+1	−1	1		**0**	**2**	**0**	**−2 → 0**	**+1**	**−1**	**+1**
5 → C	+1	−1	1		**1**	**1**	**1**	**+1 → 1**	**0**	**0**	**0**
6 → D	+1	+1	1		**1**	**1**	**1**	**+3 → 1**	**0**	**0**	**0**
7 → A	−1	−1	0		**1**	**1**	**1**	**−1 → 0**	**0**	**0**	**0**
8 → B	−1	+1	1		**1**	**1**	**1**	**+1 → 1**	**0**	**0**	**0**
. . .								**stable**			

5.2 Learning Process for Multilayer Feedforward Networks: Generalized Delta Rule or Error Backprogagation

Generalized delta rule for supervised learning:

- Same iterative procedure as for the single layer network with the following corrections during each iteration ...

  ... for the output layer with $\underline{x}_O = \underline{y}_H$:

  $$\underline{W}_{O,corrected} = \underline{W}_O + \Delta\underline{W}_O = \underline{W}_O + \eta \cdot \underline{\delta}_O \cdot \underline{x}_O^T \qquad \text{with} \quad \underline{\delta}_O = \underline{\underline{F}_O'(u_O)} \cdot (\underline{y}^* - \underline{y})$$

  $$\underline{b}_{O,corrected} = \underline{b}_O + \Delta\underline{b}_O = \underline{b}_O + \eta \cdot \underline{\delta}_O \cdot 1$$

  $\underline{\underline{F}_O'(u_O)}$ is a m x m diagonal matrix with elements $\partial f_O(u_O)/\partial t \,|\, u_O = u_O(\underline{x})$ on the main diagonal.

  ... for the hidden layer:

  $$\underline{W}_{H,corrected} = \underline{W}_H + \Delta\underline{W}_H = \underline{W}_H + \eta \cdot \underline{\delta}_H \cdot \underline{x}^T \qquad \text{with} \quad \underline{\delta}_H = \underline{\underline{F}_H'(u_H)} \cdot \underline{W}_O^T \cdot \underline{\delta}_O$$

  $$\underline{b}_{H,corrected} = \underline{b}_H + \Delta\underline{b}_H = \underline{b}_H + \eta \cdot \underline{\delta}_H \cdot 1$$

  $\underline{\underline{F}_H'(u_H)}$ is a k x k diagonal matrix with elements $\partial f_H(u)/\partial t \,|\, u = u_H(\underline{x})$ on the main diagonal.

The output layer is corrected proportional to the error $\underline{\delta}_O \sim \underline{y}^* - \underline{y}$. The correction of the hidden layer is based on the output layer's error propagated backwards to the hidden layer.

For a mathematical explanation of the generalized delta rule see the appendix.

6. Golf player example – continued from chapter 1

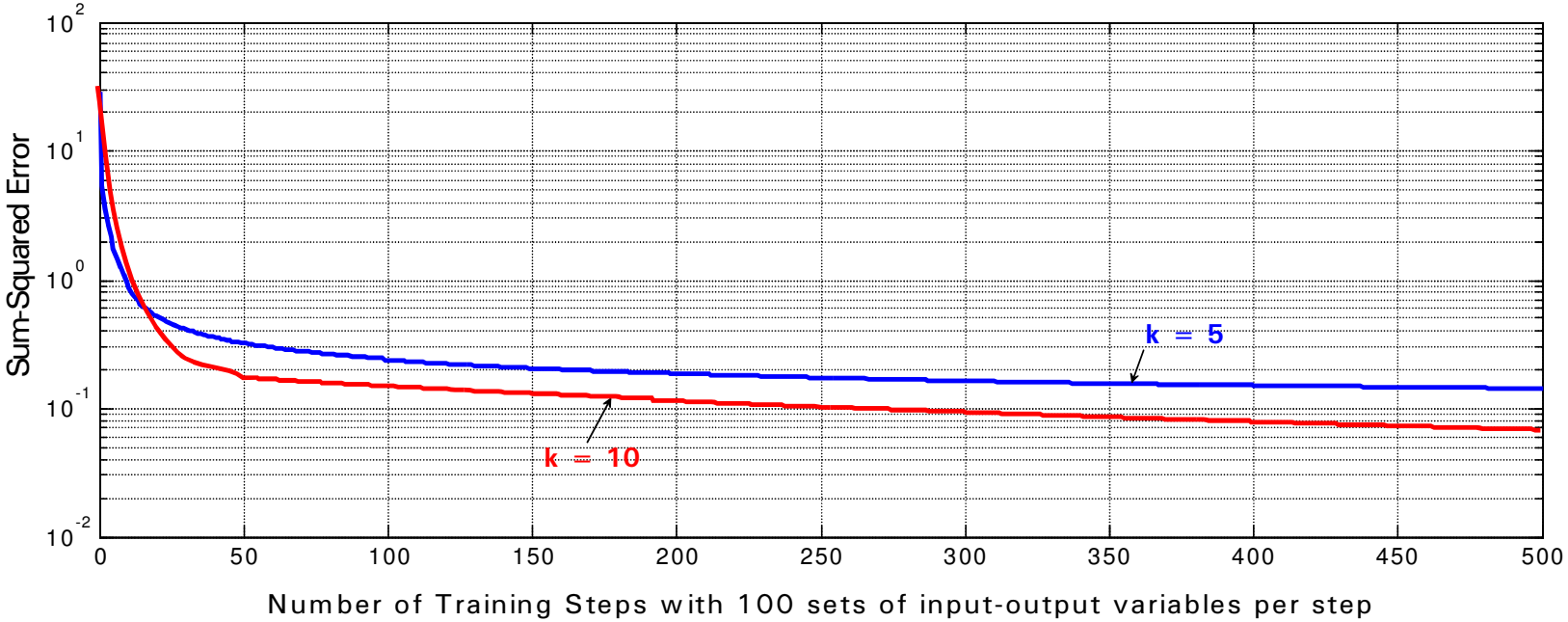
The golf player problem shall be solved by a two layer feedforward network. The inputs and outputs are predefined by the problem:

- Inputs: $\underline{e} = [\, h;\, s \,] \rightarrow n = 2$ Outputs: $\underline{y} = [\, \alpha;\, v \,] \rightarrow m = 2$

The number of neurons in the hidden layer k can be chosen freely. Two values were tried:

- $k = 5$ and $k = 10$

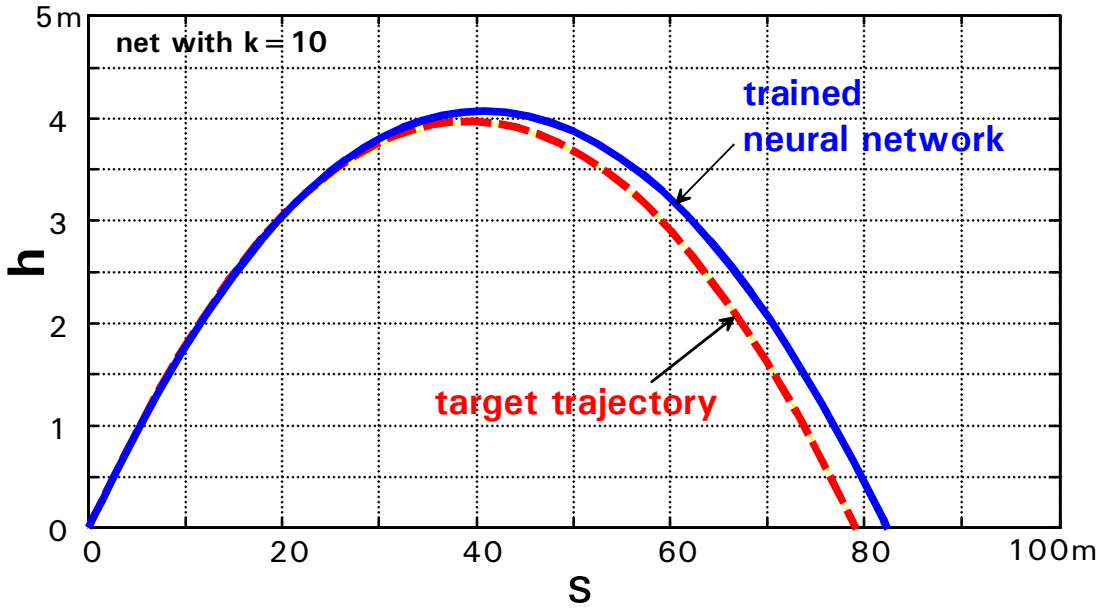
The two networks were trained using the ideal control algorithm derived from eq. (1) and (2) and the sum of the squared error of y was plotted during the training process:



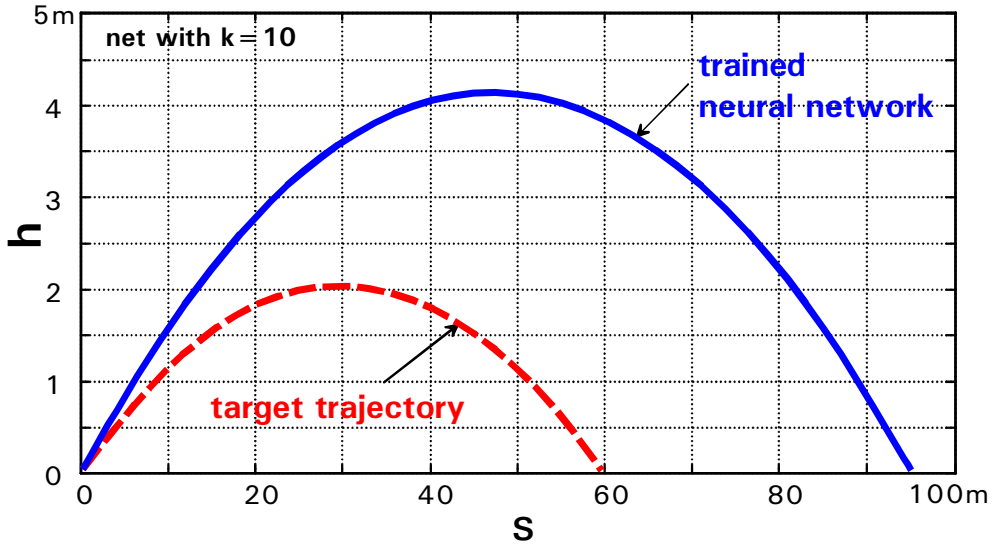
We note the following behavior, which is typical for feedforward networks:

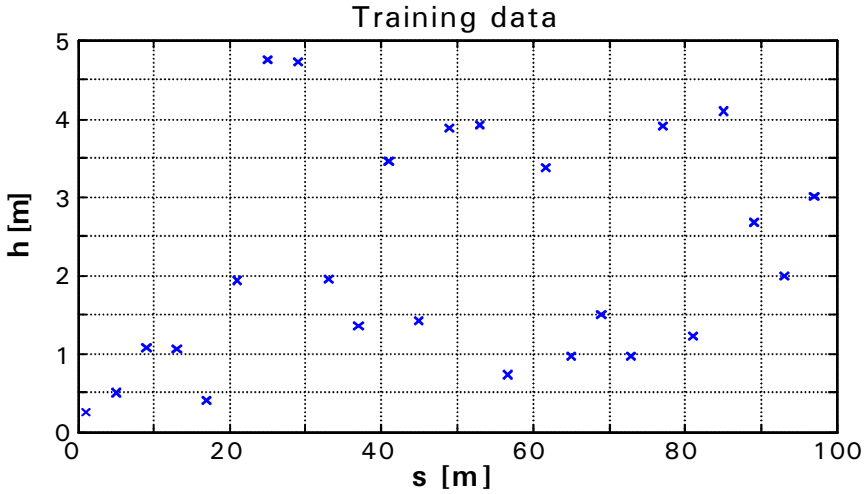
- The error at the beginning of the training process depends on the initial mismatch of the weight and bias factors.
- The network learns very fast at the beginning of the training process (here up to ~50 steps), then the error improvement slows down.
- At the end of the training process the network with more neurons in the hidden layer shows a smaller error (however it is much more complicated in realization).

To test the neural network, we let the net 'play' golf with s = 80m, h = 4m :



The result is not perfect, but pretty good. However, trying with s = 60m, h = 2m shows, that the net does not perform well in all operating points . . .





The reason of the problems are clear, when we look at the training data. The training set consisted of 25 (s, h) pairs, which did not cover the complete operating range and has big gaps. The net will work much better for data sets, which are close to those, for which it has been trained.

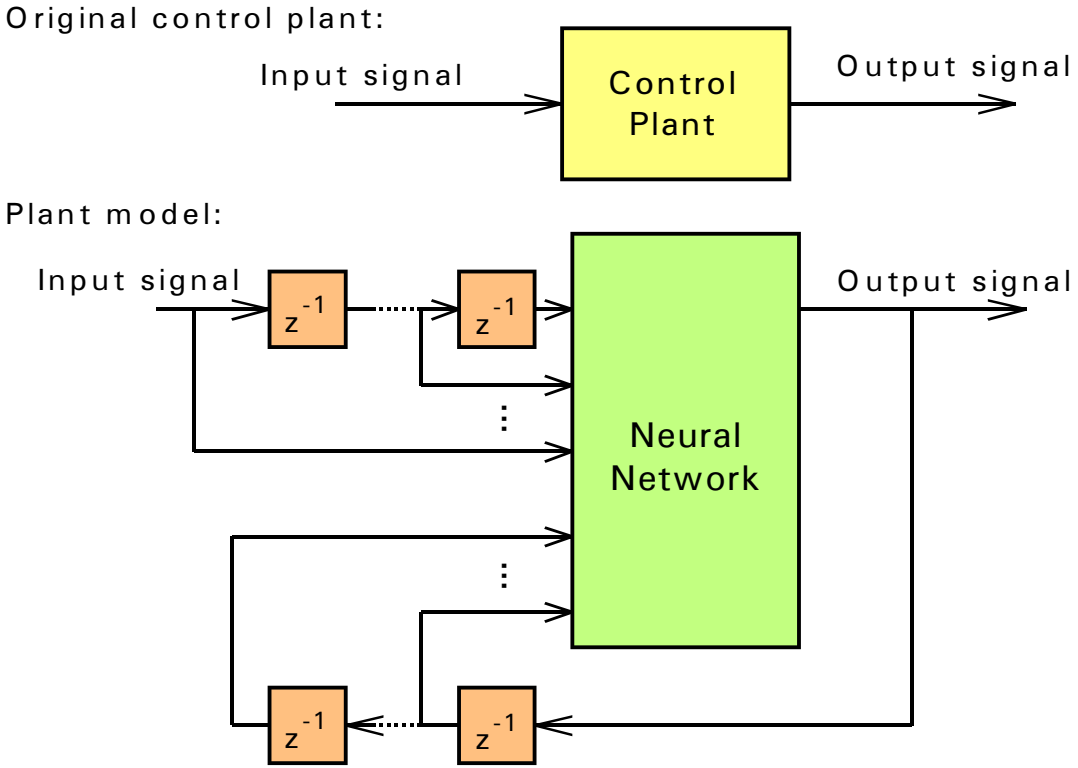
Some remarks about training:

- Input and output **signals should be normalized** to avoid numerical problems with weight factors with different orders of magnitude
- **Use** any **available knowledge** about input – output relations **to select initial values** for the weight and bias factors.
- The set of input – output variables used for training should cover the complete operating range. Neuronal **networks** are **good in interpolation, but bad in extrapolation**.
- Remove any outliers in the training data.
- **Don't overtrain** the network. When the network is trained too long, it may work perfectly for the set of training data, but may fail with other data. To test the quality of the trained network, **use another data set for verification**.
- The number of network inputs and outputs normally is predefined by the problem to be solved. However, the **number of neurons in the hidden layer** can be chosen freely:
    - **If** the **trained network shows good results** (=small errors) during verification, **try to reduce** the **number of neurons in the hidden layer** to make the network as simple as possible.
    - **If** the **trained network does not show good results** during verification, **try to increase the number of neurons in the hidden layer**.
- Software packages like Matlab's neural network toolbox come with **more advanced training methods** (e.g. adaptive learning rate) which do converge faster than standard backpropagation.

7. Applying Neural Networks in Control Engineering

7.1 Control Plant Models

For controller design and in observers of state space controllers models of the control plant are needed. If the plant is heavily nonlinear or if it is difficult or impossible to mathematically describe the dynamic behavior of the plant (e.g. in many chemical processes), a neural network can be used as a plant model:



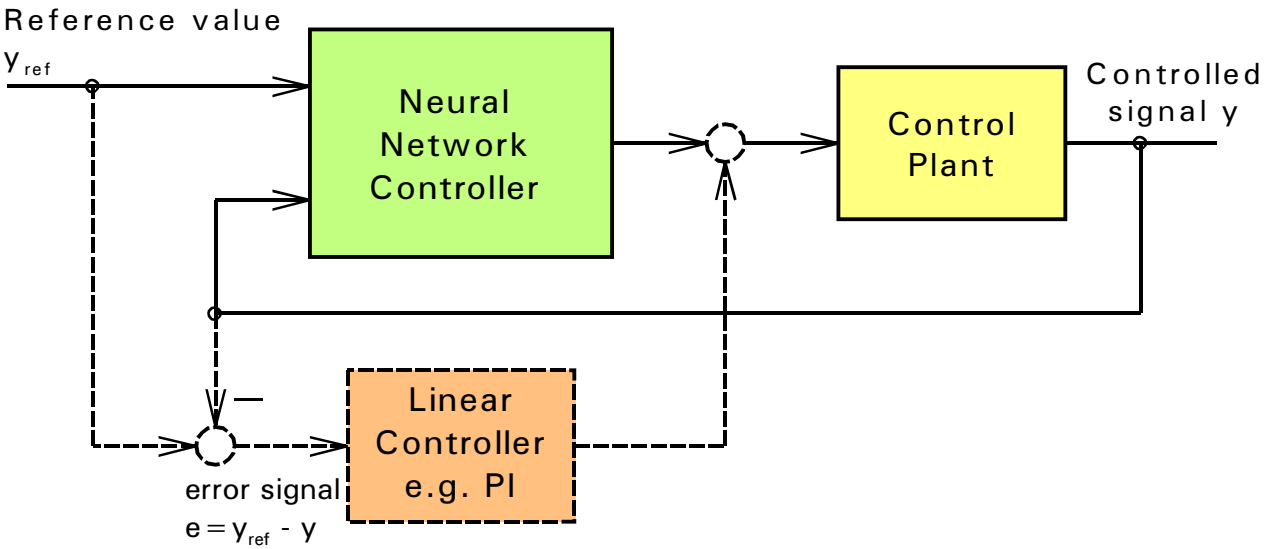
The input signal and/or the output signal is/are fed into the network via **unit delay chains** to allow the network to model the dynamic behavior of the plant.

The number of delay elements must be estimated.

Training data for the network is generated by **applying time signals to** the input of to **the real plant and** by **measuring its output signal**.

7.2 Nonlinear Controllers

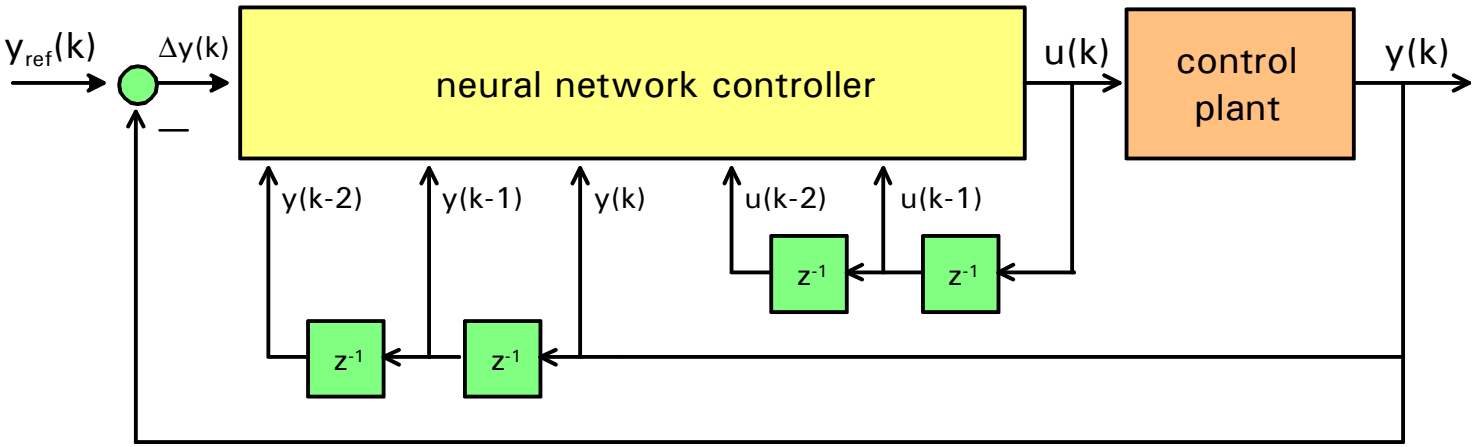
A two layer feedforward network can be used as a nonlinear controller:



- A classical linear controller (or state-space controller) can be connected in parallel to the neural network to simplify the learning process or to eliminate steady state errors.
- As with classical controllers, additionally feeding the network with the time derivate $\frac{de}{dt}$ or the time integral $\int e\, dt$ of the error signal can improve the dynamic behavior.
- Training is done by applying reference signals and modifying the network parameters to minimize the control error.

7.3 Neural Network as Inverse Reference Model Controller

Closed loop control system (example):

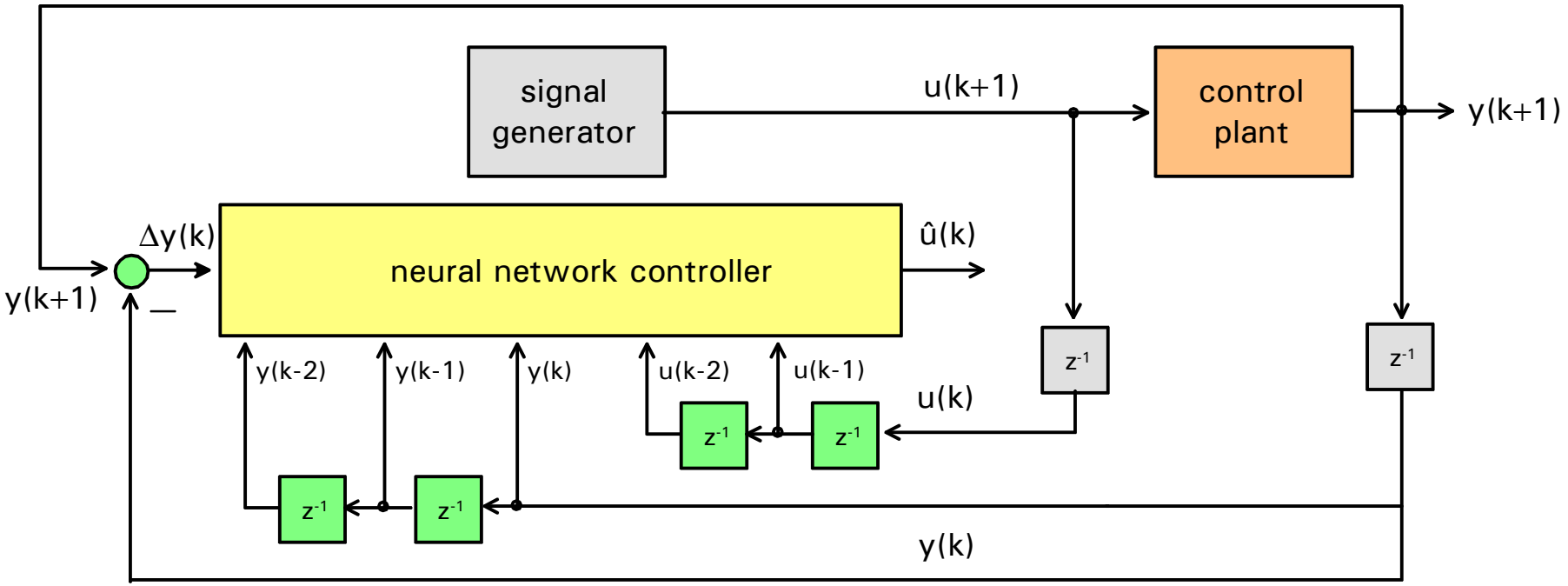


A neural network shall be used as controller for a control plant. The neural network shall generate the actuator signal u and is fed by the control error $\Delta y = y_{ref} - y$. Additionally the delayed controlled signal y(k-1), y(k-2) and the delayed actuator signal u(k-1), u(k-2) are fed into the network.

The network was trained to predict the actuator signal u(k) based on the control error $\Delta y(k)$ and the past values of y and u.

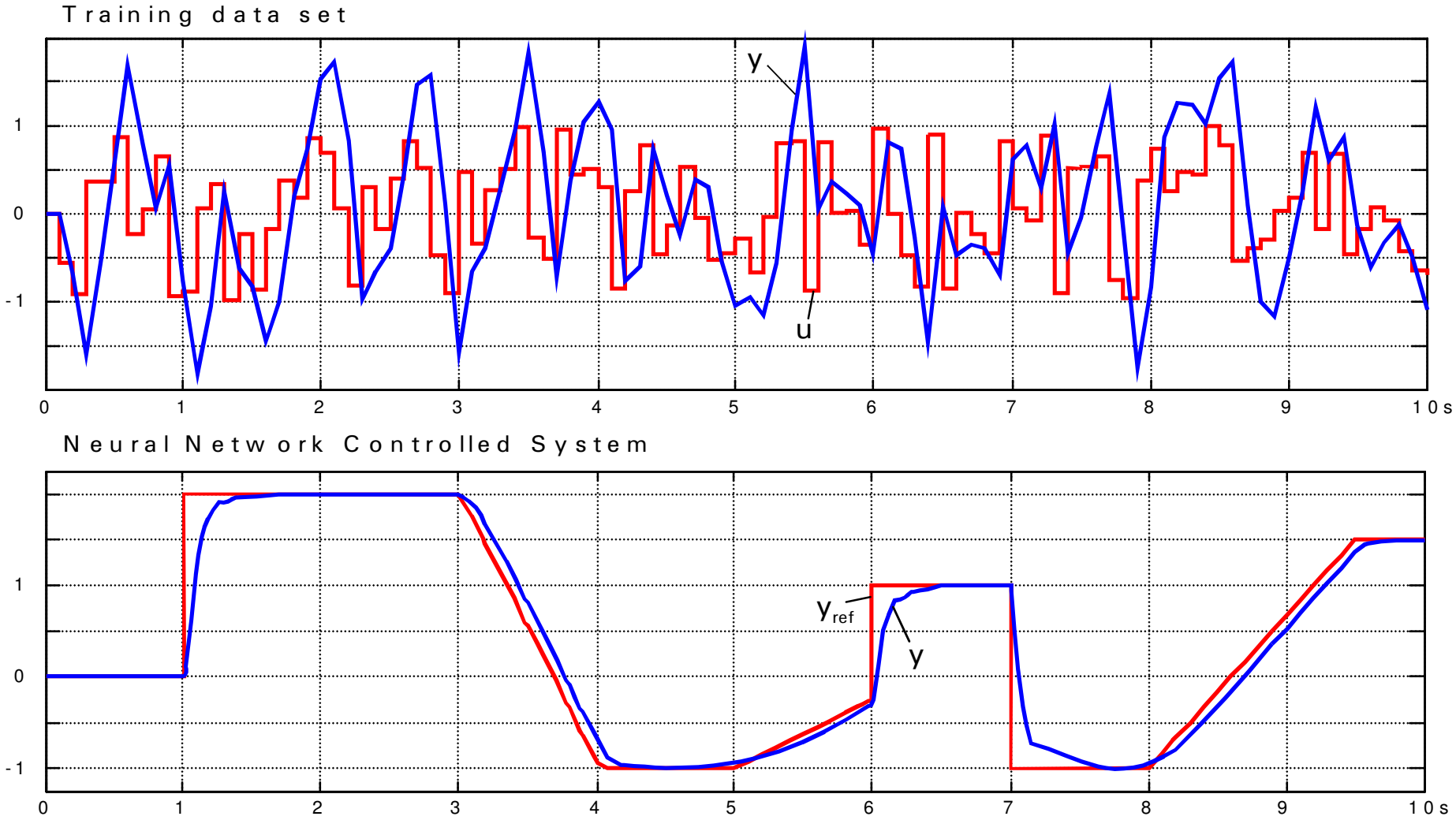
Training arrangement

During training the network learns, which sequence of actuator signals u(k), u(k-1), u(k-2) lead to the output signal sequence y(k+1), y(k-1), y(k-2), i.e. the network learns to look on sample step ahead, i.e. it learns to minimize û(k)–u(k).



The advantage of using the neural network here is, that the dynamic behavior of the control plant need not be mathematically modeled, as the neural network learns the plant's dynamics from measured data.

For a practical example see Matlab file NNControl.m on my web page:



Literature:

[1] M. Hagan, H. Demuth, M. Beale: Neural Network Design.
http://hagan.ecen.ceat.okstate.edu/ nnd.html

These authors have written an excellent, but very expensive text book. Some of the book's material, especially slides from Prof. Hagan's lectures, are available under the above link. The same people have also written Matlab's Neural Network Toolbox. Part of the User Guide is identical with the book.

[2] H.Demuth, M.Beale:
Neural Network Toolbox. User's Guide for the Matlab Toolbox.
Math Works Inc., see the Matlab online documentation

[3] M. Mokhtari, M. Marie: Engineering Applications of Matlab and Simulink. Springer Verlag.

The book contains some theory and a lot of practical examples for all types of engineering problems. Unfortunately, the book was written in French and later translated into English. In my opinion, the technical contents is easier to understand than their English ... So, if your French is good enough, maybe you'd better buy the French original.

[4] R. Rojas: Neural Networks – A Systematic Introduction. Springer Verlag. http://page.mi.fu-berlin.de/~rojas/neural

There are a lot of books and texts about neural networks available in the FHTE library, bookstores and on the internet. However, neural networks is still a domain with heavy research ("work in progress"). A lot of books are very mathematically oriented, require sound knowledge of linear algebra and optimization. Very often they either deal with theoretical aspects without linking them to practical application or they report about practical applications assuming, that the reader is already familiar with the necessary theory . . .

Appendix A: Mathematical explanation of the learning rules

A.1 Delta learning rule for single layer feedforward networks

- In a training step an input vector $\underline{x}$ is applied expecting an output vector $\underline{y}^*$
- The networks real output vector is $\underline{y} = \underline{f}(u)$ with $\underline{u} = \underline{\underline{W}}\,\underline{x} + \underline{b}$
- So the error vector in this training step is $\underline{e} = \underline{y}^* - \underline{y}$
- The weight and bias factors shall be modified in order to minimize the sum of the squared errors (a scalar value rather than a vector). The sum of the squared errors can be written as

$$E = \frac{1}{2}\sum_{i=0}^{m}(y_i^* - y_i)^2 = \frac{1}{2}\,\underline{e}^T\,\underline{e} = \frac{1}{2}\left(\underline{y}^{*T}\cdot\underline{y}^* + \underline{y}^T\cdot\underline{y} - 2\,\underline{y}^T\cdot\underline{y}^*\right)$$

- The gradient of the sum E of the squared errors with respect to the weight matrix $\underline{\underline{W}}$ is

$$\nabla E_{|W} = \frac{\partial E}{\partial \underline{\underline{W}}} = \frac{\partial E}{\partial \underline{u}}\,\frac{\partial \underline{u}}{\partial \underline{\underline{W}}}$$

with $\dfrac{\partial \underline{u}}{\partial \underline{\underline{W}}} = \underline{x}$ m x 1 vector

and $\underline{\delta} = -\dfrac{\partial E}{\partial \underline{u}} = -\dfrac{\partial E}{\partial \underline{y}}\cdot\dfrac{\partial \underline{y}}{\partial u} = \underline{\underline{F'(u)}}\cdot(\underline{y}^* - \underline{y})$ m x 1 vector

where $\underline{\underline{F'(u)}}$ is a m x m diagonal matrix with diagonal elements $\dfrac{\partial \underline{y}}{\partial u} = \dfrac{\partial \underline{f}(u)}{\partial u}\Big|_{u=u(\underline{x})}$

i.e. the derivative of the activation functions at the current operating point $u(\underline{x})$

- The correction is done proportional to the gradient (direction of the steepest descent)

$$\underline{W}_{corrected} = \underline{W} - \eta \cdot \nabla E_{|W} = \underline{W} + \eta \cdot \underline{\delta} \cdot \underline{x}^T = \underline{W} + \eta \cdot \underline{\underline{F'(u)}} \cdot (\underline{y}^* - \underline{y}) \cdot \underline{x}^T$$

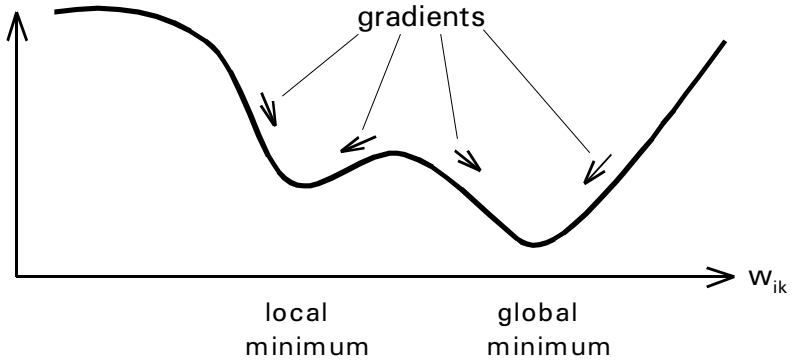
Notes: If $\frac{\partial f(u)}{\partial u}$ can not be computed, e.g. with hard limiting activation functions, set $\frac{\partial f(u)}{\partial u}$ to one.

$\underline{\delta}$ and $\underline{y}^* - \underline{y}$ are m x 1 vectors, $\underline{\underline{F'(u)}}$ is a m x m diagonal matrix, i.e. all elements outside the main diagonal are 0. Thus for the element i, k this leads to $W_{i,k,corrected} = W_{i,k} + \eta \cdot \frac{\partial f(u)}{\partial u}\Big|_{u=u(\underline{x})} \cdot (y_i^* - y_i) \cdot x_k$

Method of steepest descent:

Sum of squared errors

$$E = \sum_m (\underline{y}^* - \underline{y})^2$$



As always with gradient based methods, convergence may be slow or only a local minimum instead of a global minimum will be reached. In computer based mathematical optimization packages like Matlab more advanced algorithms, basically modified gradient based methods, exist to overcome these problems.

- Doing the same with b leads to

$$\underline{b}_{corrected} = \underline{b} - \eta \cdot \nabla E_{|b} = \underline{b} + \eta \cdot \underline{\delta} = \underline{b} + \eta \cdot \underline{\underline{F'(u)}} \cdot (\underline{y}^* - \underline{y})$$

A.2 Backpropagation algorithm for multilayer feedforward networks

- For the output layer we have exactly the same situation as with a single layer network (except some variable renaming):

$$\underline{\underline{W}}_{O,corrected} = \underline{\underline{W}}_O - \eta \cdot \nabla E_{|wO} = \underline{\underline{W}}_O + \eta \cdot \underline{\delta}_O \cdot \underline{x}_O^T$$

with $\underline{x}_O = \underline{y}_H = \underline{f}_H(u_H)$, $\underline{u}_H = \underline{\underline{W}}_H \cdot \underline{x} + \underline{b}_H$

and $$\underline{\delta}_O = -\frac{\partial E}{\partial \underline{u}_O} = -\frac{\partial E}{\partial \underline{y}} \cdot \frac{\partial \underline{y}}{\partial u_O} = \underline{\underline{F_O'(u)}} \cdot (\underline{y}^* - \underline{y})$$ with $\underline{\underline{F_O'(u)}} = \frac{\partial \underline{f}_O(u)}{\partial u}\Big|_{u_O = u(\underline{x})}$

- For the hidden layer we have:

$$\underline{\underline{W}}_{H,corrected} = \underline{\underline{W}}_H - \eta \cdot \nabla E_{|wH} = \underline{\underline{W}}_H + \eta \cdot \underline{\delta}_H \cdot \underline{x}^T$$

with $$\underline{\delta}_H = -\frac{\partial E}{\partial \underline{u}_H} = -\frac{\partial E}{\partial \underline{u}_O} \cdot \frac{\partial \underline{u}_O}{\partial \underline{y}_H} \cdot \frac{\partial \underline{y}_H}{\partial u_H} = \underline{\underline{F_H'(u)}} \cdot \underline{\underline{W}}_O^T \cdot \underline{\delta}_O$$ with $\underline{\underline{F_H'(u)}} = \frac{\partial \underline{f}_H(u)}{\partial u}\Big|_{u_H = u(\underline{x})}$

Comparing the results for $\underline{\delta}_O$ and $\underline{\delta}_H$ we see, that the error $\underline{\delta}_O$ at the output layer is propagated backwards to the hidden layer by the term $\underline{\underline{F_H'(u)}} \cdot \underline{\underline{W}}_O^T$

Using Matlab's Neuronal Network Toolbox

This chapter is only included in the online version of this manuscript.

Appendix B: Using Matlab's Neural Network Toolbox

Here only the most basic command are described to get you started. For detailed information see the Neural Network Toolbox User's Guide available in HTML or in PDF-format in Matlab's online documentation. To access it, in a Matlab command window type **`doc nnet`**. For a short overview type **`help nnet`**. For help with a certain command use help **`<command>`**.

B.1 Creating Networks

Create a Perceptron network, i.e. a single layer feedforward network with a hardlimiting activation function (in Matlab: 'hardlim')		**`net = newp(minmax, nNeurons);`**
Parameters:	**`nNeurons`**	number of neurons
	`minmax`	matrix with minimum and maximul values of the inputs: **`[ x1min x1max ;`** **`x2min x2max ;`** `. . .  ]`** These values are used for normalization. The number of rows automatically defines the number of inputs

Create an Adalin network, i.e. a single layer feedforward network with a linear activation function (in Matlab: 'purelin')	**`net = newlin(minmax, nNeurons);`** For a parameter description see **`newp()`**.

Create a double layer feedforward network with error back-propagation learning		**`net = newff(minmax, [ nH nO ], { fH fO } );`**
Parameters:	**`minmax`**	matrix with minimum and maximul values of the inputs, see **`newp()`** for details

	nH	Number of neurons in the hidden layer
	nO	Number of neurons in the ouput layer
	fH	Activation function of the hidden layer neurons
	fO	Activation function of the output layer neurons Use one of **'purelin'**, **'tansig'** or **'logsig'**. Please note, that the activation functions are contained in **'...'** and the complete list is contained in curly braces **{ ... }**

As with most Matlab functions there is a bundle of optional parameters which can be used to fine tune the network. To get you started, normally default values are a good choice.

B2. Properties of the neural network object

Matlab manages neural networks as objects with a set of properties. To see, which properties a network has, simple type the name of the network, e.g. **net**. The most important properties are:

net.inputs{1}.size	Number of network inputs
net.numLayers	Number of layers
net.layers{i}.size	Number of neurons in layer i (i = 1 input layer)
net.IW{1}	Matrix of weight factors in the input layer
net.LW{2}	Matrix of weight factors in the output layer (for double layer networks only)
net.b{i}	Vector of bias factors for the layer i (i = 1 input layer)
net.trainParam.epochs	Maximum number of 'epochs' used when training the network. During a training 'epoch' each set of input variables is applied once and correction values for the weight and bias factors are computed. This procedure is repeated 'epochs' times.
net.trainParam.goal	Goal for the sum squared error. Network training will be stopped, when either the error goal or the maximum number of epochs is reached.

Please note, many of the properties use Matlab's data type cell array. Cell arrays are some sort of super matrices, where each cell array element can contain elements of different data types and sizes. To access a cell array element, you must use curly braces **{...}** to index the cell array. If the cell array element itself is a normal matrix, to access one of the matrix elements, you must use round round braces **(...)** to index the matrix, e.g. to access the element in the 2nd row, 4th column of the output layer weight matrix of a double layer feedforward net, use **net.LW{2}(2,4)**. To display a complete cell array, use **celldisp(…)**.

B3. Training of the Neural Network

Learning procedure to adapt weight and bias factors. (Initial values will be selected automatically)		**net.trainParam.epochs=500;** //optional, default: 100 **net = train(net, x, yS);**
Parameters:	**net**	The neural network as created in B.1
	x	Input variable training sets in the following format: **[x11 x12 x13 . . . ;** **x21 x22 x23 . . . ;** **. . .]** Each column contains one set of input signals
	yS	Output variable training sets. Same format as **x**.

B3. Testing the Neural Network

Compute the networks output variables for one (or more) sets of input variables.		**y = sim(net, x);**
Parameters:	**net**	The neural network as created in B.1
	x, y	Format described in B.3

Matlab files for the orange - banana sorting net (Fruit1.m) and the golf playing net (Golf.m), demonstrating how to use these toolbox functions, can be found on the web site for my lecture.

The Neural Network Toolbox also comes with a graphical user interface to help design networks. To start it in a Matlab command window type **`nntool`**. For a tutorial how to use it, see the toolbox user's guide, chapter 3. Perceptrons, GUI (pg. 3-23 in the PDF-version).

The toolbox also comes with several examples. Some interesting ones are:

`nnd2n1`	Demonstrate several activation functions.
`nnd4pr`	Demonstrate perceptron learning (single layer feedforward net)
`nnd11bc`	Demonstrate error backpropagation learning (double layer feedforward net)

Click on the 'Contents' button in one of the demos to see a complete list of available demos.

Please note:

Matlab unfortunately names activation functions as 'transfer function', which is quite confusing, because this is a static non-linear function, whereas 'transfer function' in control engineering is the name for linear Laplace s-domain or z-domain functions, describing the dynamic characteristics of linear systems.