Neural networks

doc. RNDr. Iveta Mrázová, CSc.

Department of Theoretical Computer Science and Mathematical Logic

Faculty of Mathematics and Physics

Charles University in Prague

Neural networks

– Multi-layered neural networks– – analysis of their properties –

doc. RNDr. Iveta Mrázová, CSc.

Department of Theoretical Computer Science and Mathematical Logic

Faculty of Mathematics and Physics

Charles University in Prague

Kolmogorov´s theorem - 1957

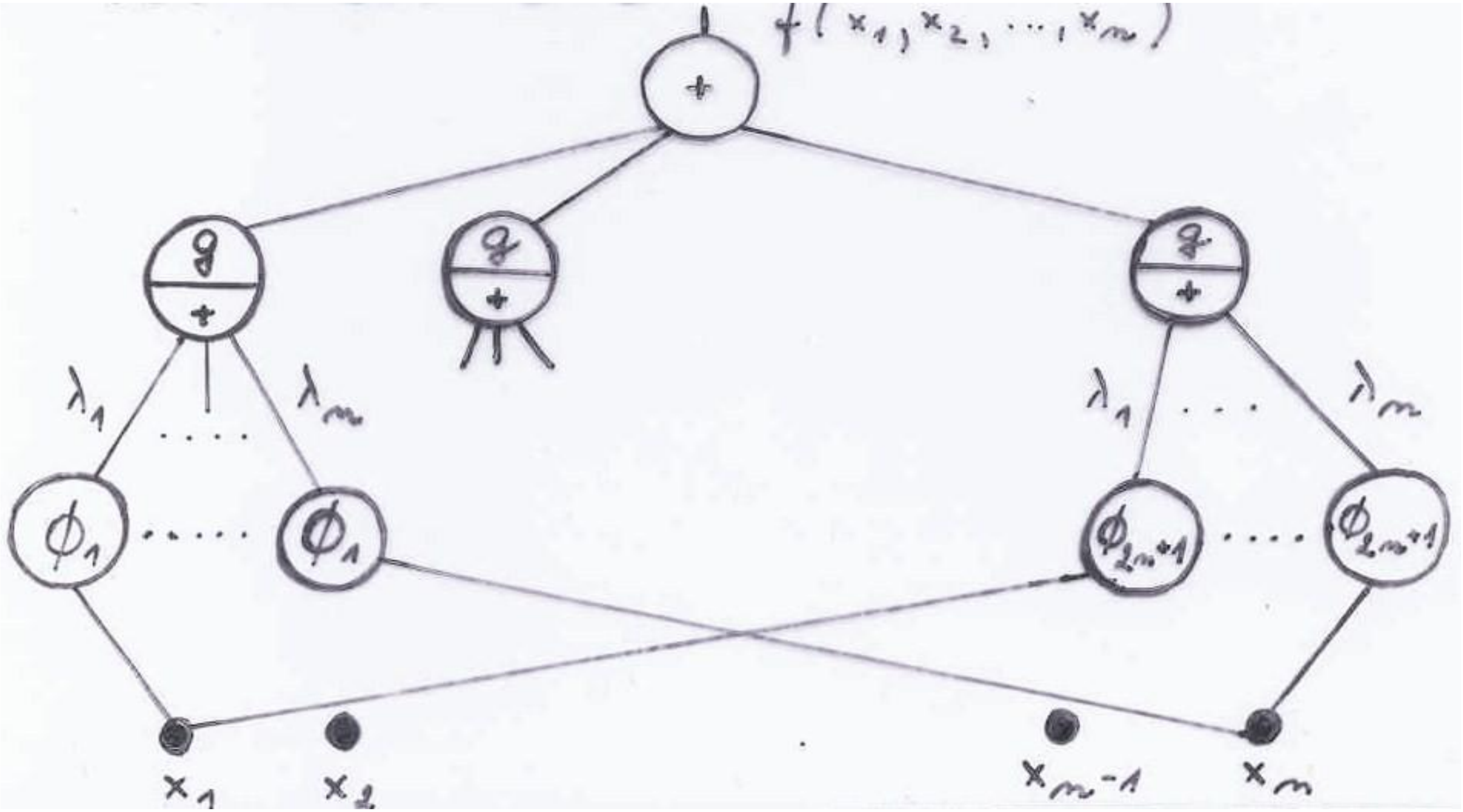
13. Hilbert problem ~ continuous functions of ***n*** arguments can always be represented using a finite composition of functions of a single argument, and addition

- Example: $x \cdot y = exp\,(\ln x + \ln y)$

V: Let $f : [0, 1]^n \rightarrow [0, 1]$ be a continuous function. There exist functions of one argument g and Φ_q, for $q = 1, \ldots, 2n+1$ and constants λ_p, for $p = 1, \ldots, n$ such that

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} g\left(\sum_{p=1}^{n} \lambda_p \Phi_q (x_p) \right)$$

Kolmogorov networks



Function approximation (1)

- Any continuous function can be reproduced exactly by a finite network of computing units, whereby the necessary primitive functions for each node exist (× the choice of the right transfer function)
- The best possible approximation to a given function (× the choice of the right number of computating units with the considered transfer function)

Function approximation (2)

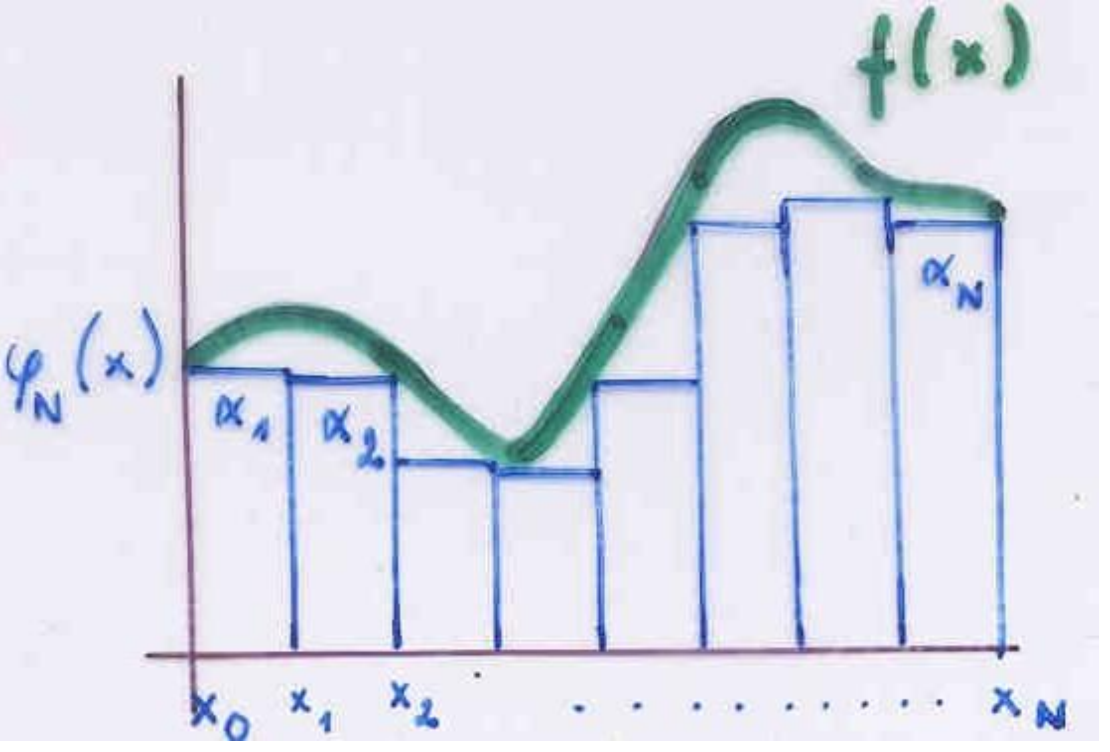
T: A cotinuous real function $f : [\,0\,,\,1\,] \rightarrow [\,0\,,\,1\,]$ can be approximated using a network of threshold elements in such a way that the total approximation error E is lower than any given real number $\varepsilon > 0$:

$$E = \int_0^1 \left| \, f(x) - \tilde{f}(x) \right| \, dx < \varepsilon$$

where $\tilde{f}$ denotes the network function.

Function approximation (3)

Proof: Idea ~ approximation of $\boldsymbol{f}$ by means of $\boldsymbol{\varphi_N}$



Function approximation (4)

Proof (continued):

- Divide the interval $[0, 1]$ into N equal segments selecting the points $x_0, x_1, \ldots, x_N \in [0, 1]$; $x_0 = 0, x_N = 1$
- Define a function φ_N as it follows:

$$\varphi_N(x) = \min\{ f(x'); x' \in [x_i, x_{i+1}) \text{ pro } x_i \le x < x_{i+1} \}$$

- Further, consider φ_N an approximation of f so that the approximation error E_N is given by:

$$E_N = \int_0^1 | f(x) - \varphi_N(x) | \, dx$$

Function approximation (5)

Proof (continued):

- Since $f(x) \geq \varphi_N(x) \quad \forall\, x \in [0, 1]$, E_N corresponds to

$$E_N = \int_0^1 f(x)\, dx - \int_0^1 \varphi_N(x)\, dx$$

~ lower Riemann sum
of the function f

- Since continuous functions are integrable $\rightarrow$ the lower sum of f converges in the limit $N \rightarrow \infty$ to the integral of f in the intervalu $[0, 1]$
- Thus it holds $E_N \rightarrow 0$ when $N \rightarrow \infty$, hence for any real number $\varepsilon > 0$ there exists an M such that $E_N < \varepsilon \;\; \forall\, N \geq M$
- The function φ_N is therefore the desired approximation of f.

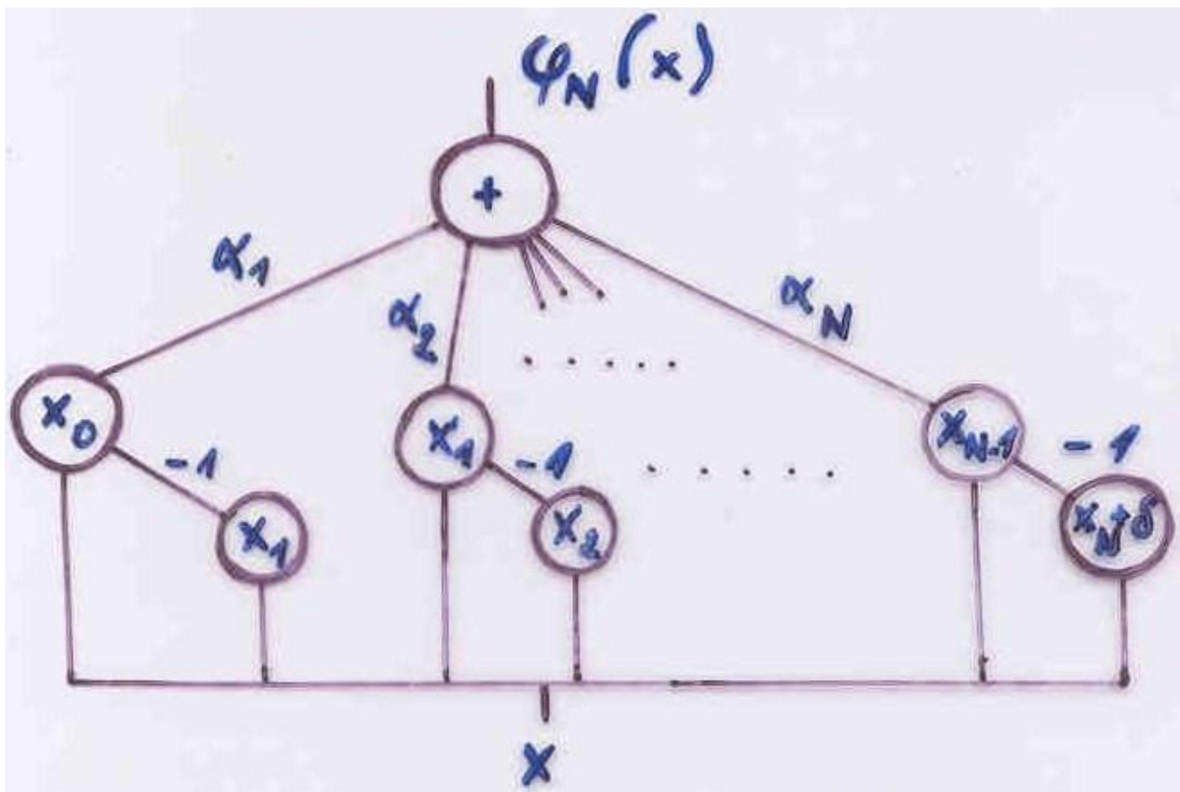
Function approximation (6)

Proof (continued):

- The function $\varphi_N(x)$ can be computed by a network of threshold units (~ neural network)
  - $\varphi_N(x)$ is a step-wise function
  - in each of the N segments of the interval $[0, 1]$: $[x_0, x_1), [x_1, x_2), \ldots, [x_{N-1}, x_N]$, $\varphi_N(x)$ has the respective value $\alpha_1, \ldots, \alpha_N$

Function approximation (7)

Proof (continued):



Function approximation (8)

Proof (continued):

- This network can compute the step-wise function $\varphi_N(x)$**:**
  - The single input to the network is x
  - Each pair of units with the weights x_i and x_{i+1} guarantees that the unit with threshold x_i will be active when $x_i \leq x < x_{i+1}$.
  - The (linear) output unit adds all outputs of the previous layer of units and produces their (weighted) sum as a result
  - The unit with the threshold $x_N+\delta$, where δ is a small positive number, is used to recognize the case $x_{N-1} \leq x \leq x_N$.
- This network computes the function φ_N, that approximates the function f with the desired maximum error.

  QED

Function approximation (9)

Corollary:

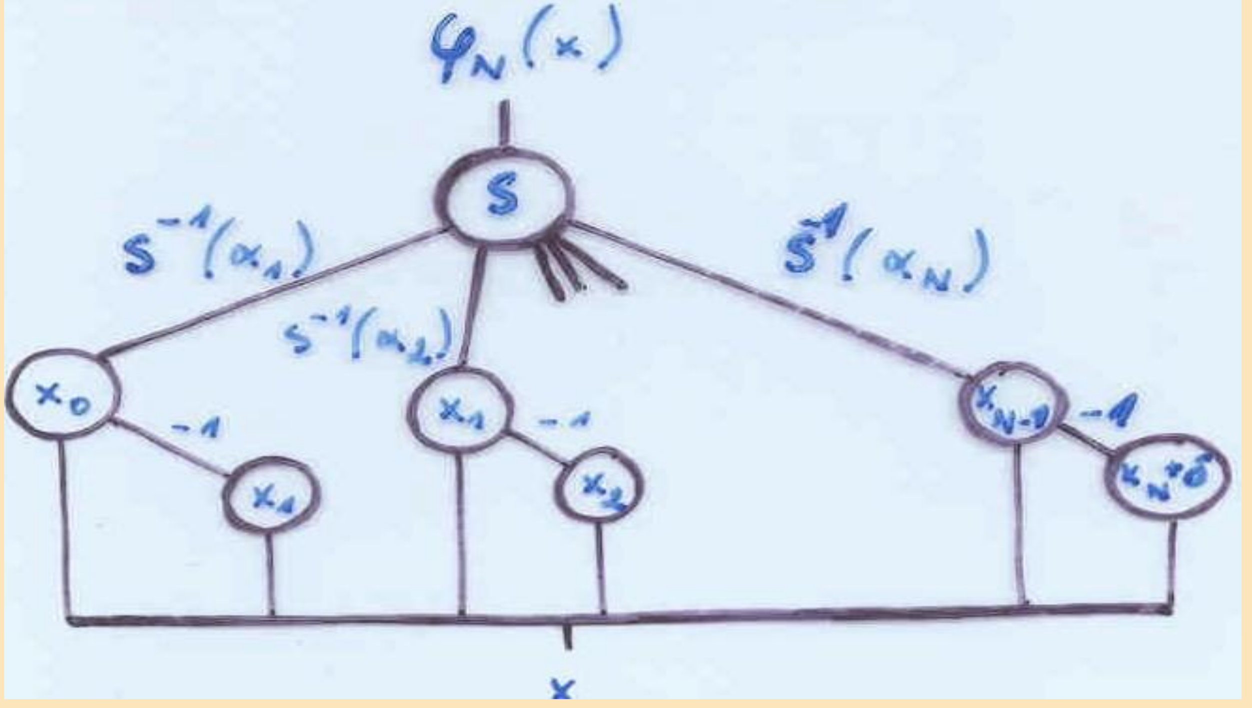
The theorem is valid also for neurons with the sig-moidal transfer function with $f : [0, 1] \rightarrow (0, 1)$

Proof:

- The image of the function f has been limited to the interval $(0, 1)$ in order to simplify the proof
- The function f can be approximated using the following network:

Function approximation (10)

Proof (continued):



Function approximation (11)

Proof (continued):

- The transfer function of the units with the threshold x_i is given by $s_c(x - x_i)$, where c controls the slope of the function

$$s_c(x - x_i) = \frac{1}{1 + e^{-c(x - x_i)}}$$

- The network can approximate the function φ_N with an approximation error lower than any desired bound (> 0)

  (~ threshold functions can be approximated with any desired precision by a parametrized sigmoidal function)

Function approximation (12)

Proof (continued):

- The weights connecting the first layer of units to the output unit have been set in such a way that the sigmoid produces the desired values α_i as a result
- Further it should be guaranteed that every input x produces a single 1 from the first layer to the output unit

  $\rightarrow$ the first layer just finds out to which of the N segments of the interval $[\,0\,,\,1\,]$ the input x belongs

QED

Function approximation (13)

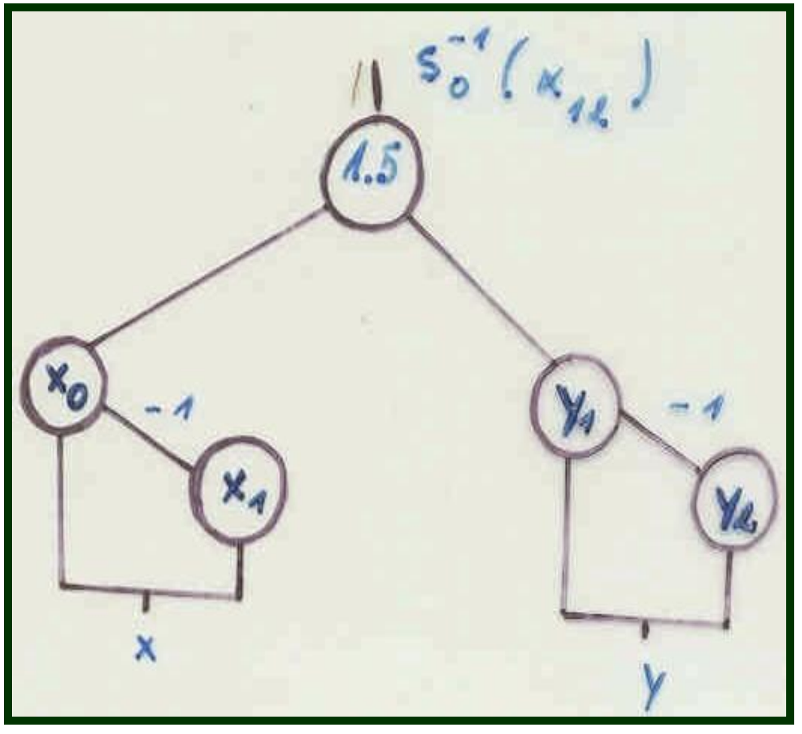
The multidimensional case:

The network capable of approximating the function $f: [0,1]^n \rightarrow (0,1)$ can be constructed using the same general idea as before in the one-dimensional case:

- extensions necessary for the two-dimensional case

- Recognition of intervals in the x and y domains
  - 2 units left are used to test $x_0 \leq x < x_1$
  - 2 units right are used to test $y_1 \leq y < y_2$
- The unit with the threshold 1.5 recognizes the conjunction of both conditions (for x and y)

Function approximation (14)



- The „output“ has the weight $\boldsymbol{s_0^{-1}(\alpha_{12})}$, so the sigmoidal transfer function yields $\boldsymbol{\alpha_{12}}$

  → this number corresponds to the desired approximation of the function $\boldsymbol{f}$ on:

$$[\,x_0, x_1\,) \times [\,y_1\,, y_2\,)$$

The complexity of learning

The satisfiability problem

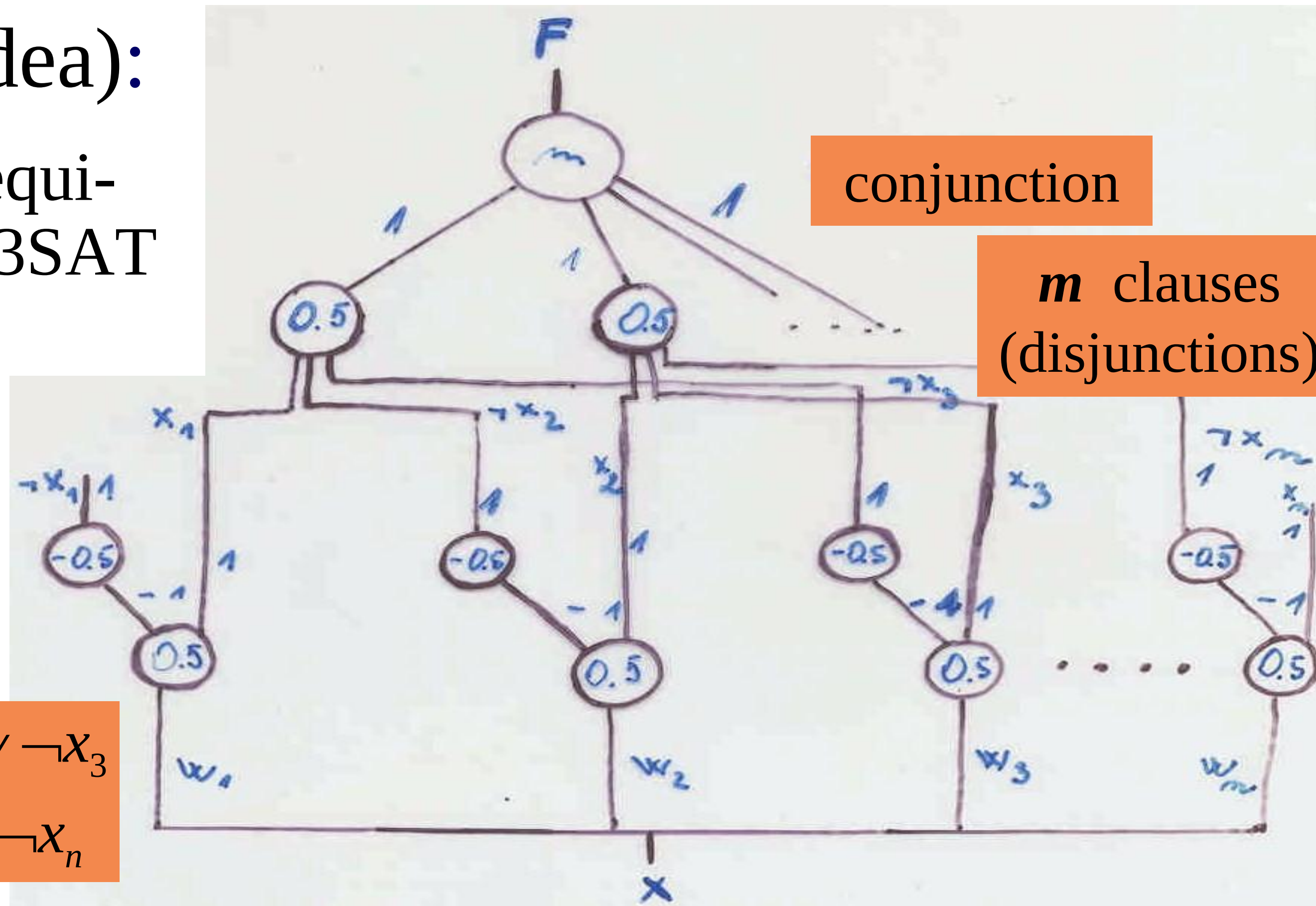
D: Let ***V*** be a set of ***n*** logical variables, and let ***F*** be a logical expression in conjunctive normal form (conjunction of disjunctions of literals) which contains only variables from ***V***.
The satisfiability problem consists in assigning truth values to the variables in ***V*** in such a way that the expression ***F*** becomes true.

T: The general learning problem for networks of threshold functions is NP-complete.

The complexity of learning (2)

Proof (idea):

network equivalent to 3SAT



conjunction

m clauses (disjunctions)

1) $x_1 \vee \neg x_2 \vee \neg x_3$

2) $x_2 \vee x_3 \vee \neg x_n$

The complexity of learning (3)

Proof (continue):

1. 3SAT can be reduced to an instance of a learning problem for neural networks in polynomial time

   A logical expression $\boldsymbol{F}$ in conjunctive normal form, which contains $\boldsymbol{n}$ variables can be transformed in polynomial time in the description of a network of the above type:

   - For each variable $\boldsymbol{x_i}$ a weight $\boldsymbol{w_i}$ is defined
   - The connections to the third layer are fixed according to the conjunctive normal form we are dealing with

The complexity of learning (4)

Proof (continue):

- This can be done (using a suitable coding) in polynomial time, because it holds for the number m of different possible disjunctions in a 3SAT formula that $m \leq (2n)^3$
- If an instantiation A with logical values of the variables x_i exists, such that F becomes true, then there exist weights $w_1, w_2, \ldots, w_n$, that solve the learning problem

The complexity of learning (5)

Proof (continue):

- It is suficient to set the weights $w_i = 1$, if $x_i = 1$; and $w_i = 0$, if $x_i = 0$.
  (in both cases, we thus choose $w_i = x_i$.)
- Similarly in the opposite way:
  if there exist weights $w_1, w_2, \ldots, w_n$, that solve the learning problem, then the instantiation $x_i = 1$ for $w_i \geq 0.5$ and $x_i = 0$ otherwise, is a valid instantiation that makes F true

The complexity of learning (6)

Proof (continue):

2. Further, we have to show that the learning problem belongs to the class NP (its solution can be checked in polynomial time)
   - If the weights $w_1, w_2, \ldots, w_n$ are given, then a single run of the network can be used to check if the output F is equal to 1
   - The number of computation steps is directly proportional to the number n of variables and to the number m of disjunctive clauses (which is bounded by the polynomial $(2n)^3$)

The complexity of learning (7)

Proof (continue):

- The time required to check an instantiation is therefore bounded by a polynomil in ***n***
- The given learning problem thus belongs to the class NP

QED

Remark:

For some special types of simple neural networks, the learning problem can be solved in polynomial time (by means of linear programming algorithms)

Number of regions in the feature space (1)

- The capacity of a neuron depends on the dimension of the weight space and the number of cuts with separating hyperplanes

→ **Question:**

How many regions are defined by $\boldsymbol{m}$ cutting hyper-planes of dimension $\boldsymbol{n-1}$ in $\boldsymbol{n}$ – dimensional space?

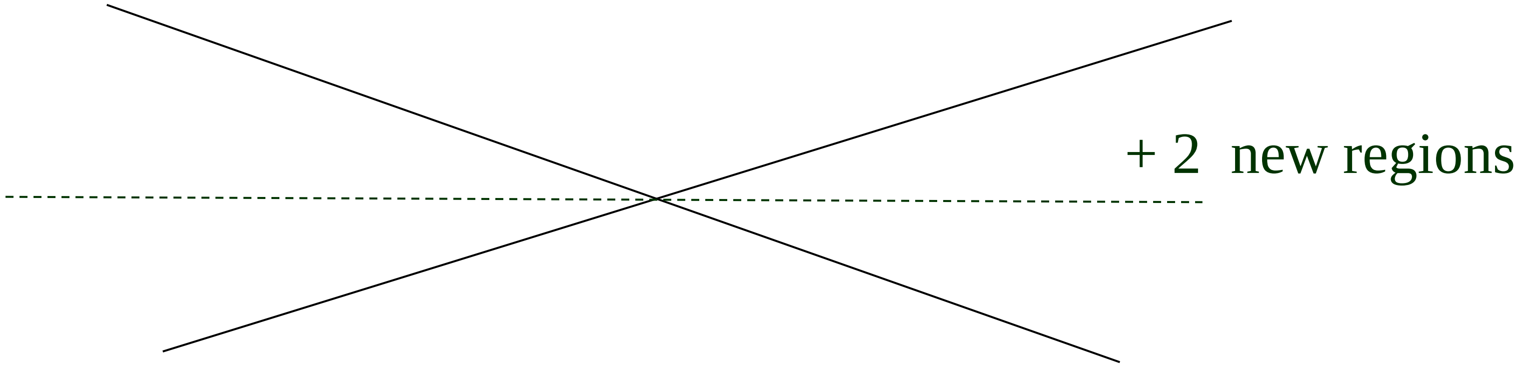
- we consider only hyperplanes going through the origin

→ Intersection of $\boldsymbol{l}$ hyperplanes; $\boldsymbol{l} \le \boldsymbol{n}$ is of dimension $\boldsymbol{n-l}$

Number of regions in the feature space (2)

- <u>2 – dimensional case:</u>

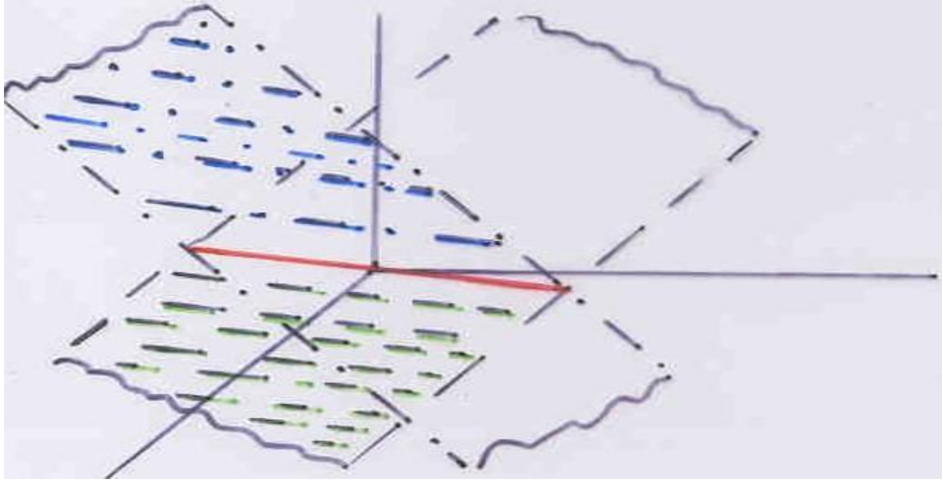
$\boldsymbol{m}$ lines going through the origin define at most $\boldsymbol{2 \cdot m}$ different regions



Number of regions in the feature space (3)

- 3 – dimensional case:

  - each new cut increases the number of regions two times



- in general: n cuts with $(n-1)$ – dimensional hyperplanes in n – dimensional space define at most 2^n different regions

Number of regions in the feature space (4)

Theorem: Let $R(m, n)$ denote the number of different regions defined by m separating hyperplanes of dimension $n-1$ in an n – dimensional space. We set $R(1, n) = 2$ for $n \geq 1$ and $R(m, 0) = 0 \quad \forall m \geq 1$.

Then for $n \geq 1$ and $m > 1$:

$$R(m, n) = R(m-1, n) + R(m-1, n-1)$$

Number of regions in the feature space (5)

Proof (by induction on m):

1. $m = 2$ and $n = 1$: The formula is valid, because
   $R(2, 1) = R(1, 1) + R(1, 0) = 2 + 0 = 2$
2. $m = 2$ and $n \geq 2$: $R(2, n) = 4$ => valid, because
   $R(2, n) = R(1, n) + R(1, n - 1) = 2 + 2 = 4$
3. $m + 1$ hyperplanes of dimension $n - 1$ are given in n-dimensional space and in general position ($n \geq 2$):
   - The first m hyperplanes define $R(m, n)$ regions in n – dimensional space

Number of regions in the feature space (6)

Proof (continue):

- $(m+1)$ – st hyperplane intersects the first m hyperplanes in m hyperplanes of dimension $n-2$
- These m hyperplanes (of dimension $n-2$) divide the $(n-1)$ - dimensional space into $R(m, n-1)$ regions
- After the cut with the hyperplane $(m+1)$, exactly $R(m, n-1)$ new regions have been created

→ The new number of regions is therefore:

$$R(m+1, n) = R(m, n) + R(m, n-1)$$

QED

Number of regions in the feature space (7)

- A useful alternative for ***R(m,n)***:

$$R(m,n) = 2 \sum_{i=0}^{n-1} \binom{m-1}{i}$$

- With a growing ***n***, the number of Boolean functions growes significantly quicker than the number of regions formed by hyperplanes in a general position

  - this number can be in general larger than the number of threshold functions over binary inputs

Number of regions in the feature space (8)

Example:

n	Nr. of Boolean functions	Nr. of threshold functions	Nr. of regions
1	4	2	2
2	16	14	14
3	256	104	128
4	65 536	1 882	3 882
5	4.3×10^9	94 572	412 736

Number of regions in the feature space (9)

Consequences:

Learnability problems ~ if the number of input vectors is too high, the network might be not able to form enough regions with the given number of hidden neurons

- **Generalization**

  ~ expected number of correctly classified examples

- **Over-fitting**

  ~ erroneous interpolation of patterns outside of the training set

- **Vapnik – Chervonenkis dimension (VC-dimension)**

  ~ finite VC-dimension $\rightarrow$ „the class of concepts“ is learnable

Vapnik – Chervonenkis dimension (VC–dimension) (1)

D: Let $C = \{ f_i \}$ be a set of functions (concept class)
The set of m training patterns $\{ t_k \}_{k=1,\ldots,m}$ **can be shattered** by means of C , if for each of the 2^m possible labelings of these patterns with $1 / 0$, these exists at least one function, that satisfies this labeling.

D: **VC-dimension** V **of a set of functions** C is defined as the biggest m , for which a set of m training patterns exists that can be shattered.

Vapnik – Chervonenkis dimension (VC–dimension) (2)

- If there exists for any ***m*** a set of ***m*** training patterns, that can be shattered by means of ***C***, the VC-dimension of ***C*** is infinite

→ Such a problem is not „ **L E A R N A B L E** “

- VC-dimension of a set of functions does not in general depend on the number of parameters
- VC-dimension impacts adequate generalization
  - The network can have many parameters, but it should have a small VC-dimension → better generalization
  - High VC-dimension correlates with worse generalization

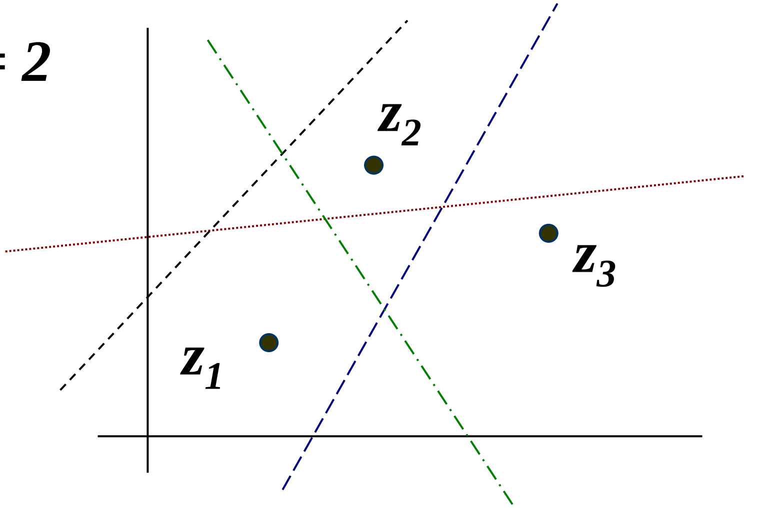
Vapnik – Chervonenkis dimension (VC–dimension) (3)

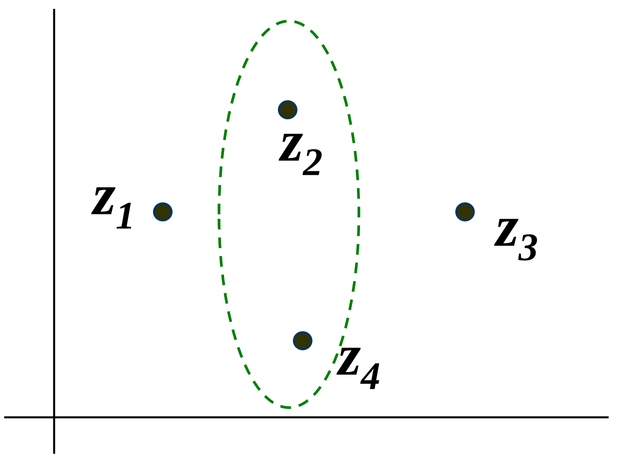
Example:

1. VC-dimension of a set of linear indicator funkctions

$$Q(\vec{z},\alpha) = \Theta\left\{\sum_{p=1}^{n} \alpha_p z_p + \alpha_0\right\}$$ in the ***n*** – dimensional space is ***n* + 1** (i.e., it can shatter at most ***n* + 1** patterns)

***n* = 2**





Vapnik – Chervonenkis dimension (VC–dimension) (4)

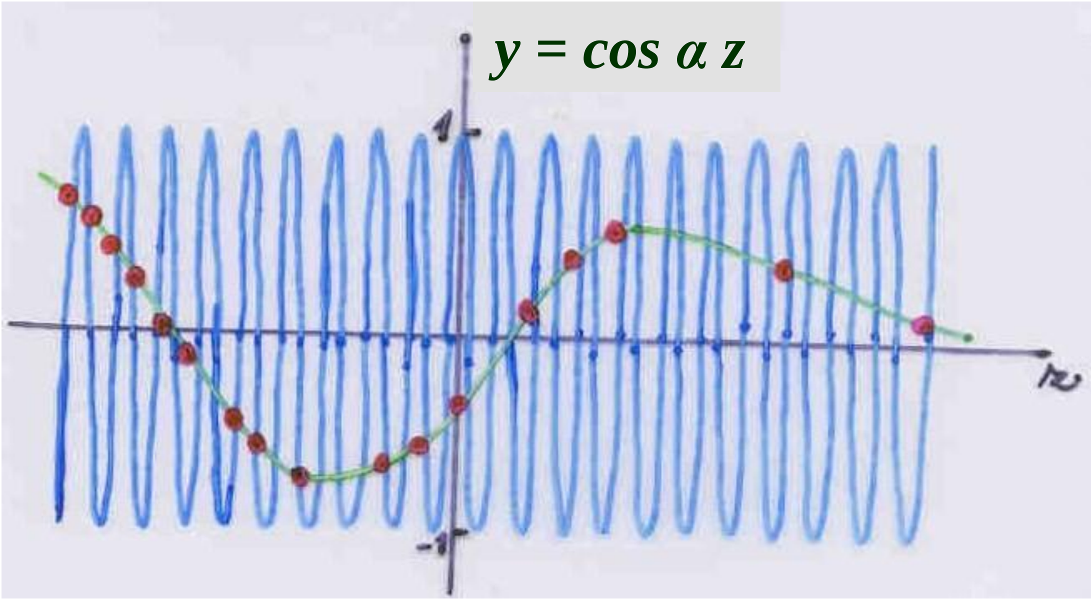
2. VC-dimension of the following set of functions
   $f(z, \alpha) = \theta(\cos \alpha z)\ ,\ \alpha \in R$ is infinite

- The points $z_1 = 10^{-1}, \dots, z_m = 10^{-m}$ can be shattered by means the functions from this set
- To shatter these patterns into two classes $(+1 / -1)$ given by the sequence $\delta_1, \dots, \delta_m ;\ \delta_i \in \{0, 1\}$ it is sufficient to choose the value of the parameter

$$\alpha = \pi \left(\sum_{i=1}^{m} (1 - \delta_i) 10^i + 1 \right)$$

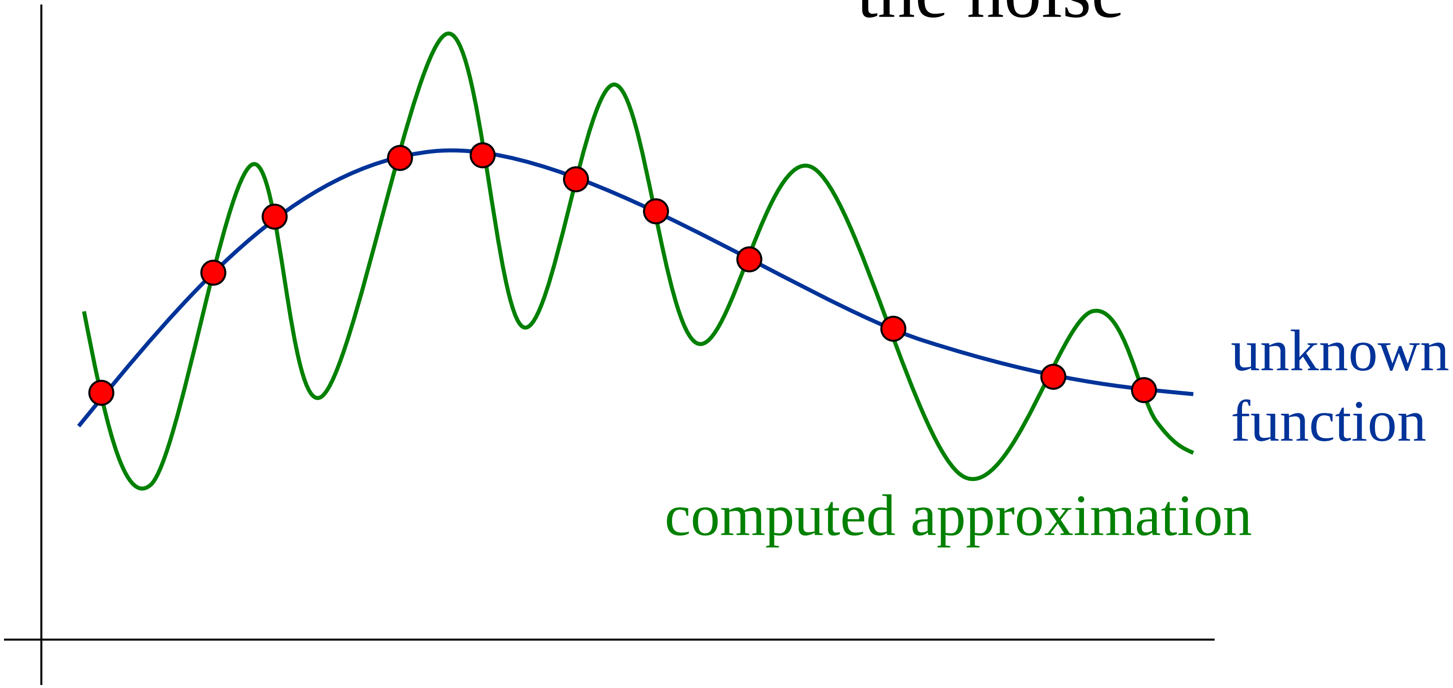
Vapnik – Chervonenkis dimension (VC–dimension) (5)

- when choosing a suitable coefficient α it is possible to approximate any function bounded in $< +1 / -1 >$ for any number m of selected points by $\cos \alpha z$



Vapnik – Chervonenkis dimension (VC–dimension) (6)

The problem of „overfitting“ ~ the network learns also the noise



Vapnik – Chervonenkis dimension (VC–dimension) (7)

- For the network with ***W*** weights and ***N*** neurons and with the required limit for the generalization error ε, the number ***P*** of training patterns necessary for good generalization is: $P \geq (W/\varepsilon) \log_2 (N/\varepsilon)$
- A multi-layered network with ***1*** hidden layer cannot generalize well, of there were less than W/ε randomly chosen training patterns, i.e., $P \geq W/\varepsilon$
  - To achieve the accuracy of at least ***90 %*** it is necessary to provide at least $10 \cdot W$ patterns