Serial decomposition of feedforward neural networks

Hubert Niewiadomski

August 1, 2003

Contents

Chapter 1

Introduction

1.1 Motivation

From the very beginning science has been developing theories that made human life easier. The main objective of science has always been a strict description of the world. In a sense this progress of knowledge gave simple answers to complex and formerly inextricable problems. I am deeply convinced that physics is the mother of our civilization and in my opinion it is important to look at artificial intelligence through the perspective of our understanding the world around. Although the physical language is hypothetical, generally not to be proved [29] and only experimentally verifiable, its assumption allowed technological development. The 19th century, the age of steam and steel, was generally based on Newton's laws of mechanics. The formulation of the relationships between magnetic and electric fields by J. C. Maxwell in 1865, began new era, the era of information. These four differential equations made the expansion of telecommunications and information technology possible and direct it until now. First analogue differentiators and integrators, based on the fact that certain classes of differential equations could be effectively simulated using electronic circuits, brought the scientists to the idea of artificial intelligence. At that time researchers suggested the simulation of brain and neurons activity with analogue elements like capacitors, coils, analogue adders and amplifiers. Obviously such structures were of limited size and could not perform complicated tasks. Quick development of electronics, invention of digital circuits, forced researchers to revise their attitude towards artificial intelligence and to express its ideas in the language of algorithms. This approach, with modifications arising from technology progress, is generally up-to-date.

Table 1.1 [79] describes computational power of stand-alone machines. Parallel and distributed computing currently offer speeds reaching $41 \cdot 10^{12}$ floating point operations per second [34, 82]. Is it enough to simulate human brain? Separate neurons are relatively slow (10^3Hz) but are intensively interconnected (even 80000 synaptic interconnections per cell). The total number of neurons in brain ($3 \cdot 10^{11}$) permanently exceeds the number of transistors even in the fastest computers. We can estimate that such simulation requires $\approx 8 \cdot 10^{18}$ Flops — unfortunately much more than the current computers can offer. Perhaps that is the reason why the brain is capable of computationally demanding perceptual acts like recognition of faces and words that are only now on the horizon of computers.

Table 1.1: Generations of computers

Generation	Years (approx.)	Technology	typical speed ops. per sec.
1	1946–1957	vacuum tube	$4 \cdot 10^4$
2	1958–1964	transistor	$2 \cdot 10^5$
3	1965–1971	small and medium scale of int.	10^6
4	1972–1977	large scale of integration	10^7
5	1978–now	very large scale of integration	10^8–10^9

FPGA reconfigurable devices, perhaps one of most promising areas of contemporary VLSI technology, give amazing possibilities for artificial neural network (ANN) implementation. Their virtual internal structure can be easily modified, and their principle elements — *logic cells*, able to perform basic multi-wire boolean and flip-flop operations — work in parallel [3, 90]. Reconfigurable computing is a new paradigm based on dynamical adapting the computation and communication structures on the chip. Reconfigurable circuits and systems have evolved from application specific accelerators to a general purpose computing paradigm. Since these devices promise a high degree of flexibility, numerous attempts have been taken to realize intelligent systems (and neural networks) on the basis of this idea [28, 87]. In comparison to computer simulations, such approach is closer to the structure of brain, because FPGA cells work in parallel — just like real neurons do. Moreover, neuronal interconnections in FPGA can be easily changed through reconfigurability.

Is artificial intelligence only a kind of algorithm or is it deeply rooted in more convolved physical theory, which may have not been discovered yet? What is the nature of consciousness? While these questions remain unanswered, they divide scientists into the supporters of so called *weak* and *strong artificial intelligence*.

We are forced to ask the question whether there are any other possible ways and models of computation than those based on deterministic algorithms and boolean logic. Maxwell's theory firmly stated that the velocity of electromagnetic waves is constant and finite. Further experiments of A. Michelson and E. Morley, as well as works of H. Lorentz, led directly to Einstein's theory of relativity (1905). The observations of micro-world forced physicists to abandon the above-mentioned continuous theories and develop a new approach – quantum mechanics (valid in micro-scale), which is governed by complex wave-functions and gives generally probabilistic results. This theory has certainly opened new areas of understanding the phenomenon of information. It suggests new rules of computation, different from our classical, deterministic approach.

According to works and ideas of D. Deutsch [22], R. Feynman [24, 25], P. Shor [75], C. H. Bennett [9], and experiments conducted for example by I. L. Chuang [52], it gives the potential possibility to reduce the time complexity of at least some NP-hard algorithms. According to R. Penrose [56] quantum processes may be involved in thinking (single graviton criterion). There also remains the land of undiscovered physics — the unification of two permanently separated theories: the general theory of relativity and quantum electrodynamics.

The above context forced me to study deeply into the abilities of contemporary artificial neural networks, especially in the context of the information measure — entropy. Interestingly enough, artificial neural network is nothing more but information channel with input and output. It produces errors (noise) and transmits converted information.

Numerous applications of ANN in medicine, experimental physics, pattern recognition, control systems, knowledge engineering, economics and other areas of human activity, show that ANN are a well established part of information science and ensure its constant future development. Why is it so? The neural networks methodology enables us to design useful nonlinear systems with large numbers of inputs, and what is probably the most important aspect, with the design based solely on instances of input-output relationships generally without any prior knowledge about the modelled system. In my opinion, one of the most necessary investigations concerning neural networks are those pertinent to their structure. There are no exact and purpose independent indications what their architecture should be like. In my work I will try to address this problem using the approach that proved very useful in logic synthesis.

1.2 Objectives of the work

The main goals of the work are:

- To provide definitions and properties of measures of information and relate them to neural network supervised training and modular systems.
- To provide the description of the structure of feedforward neural networks and their computational capabilities in the context of PAC model.
- To relate the entropy of a neural network input to its computational capabilities.
- To provide methods of designing modular neural networks.
- To test and evaluate the proposed approach, basing on real-life examples.
- To indicate further investigations concerning the approach suggested in this work.

1.3 Thesis

Currently we lack the general approach to the decomposition of neural networks. The methods based on simple statistical dependencies very often prove unsatisfactory, since statistical correlation is not a good measure of informational dependencies (Sections 2.1, 5.1). I am convinced that the decomposition of a monolithic system should be carried out on the basis of information flow between the components of the decomposed system. For that purpose I introduce Shannon's entropy as a measure of information (Sections 2.2–2.5) and show the entropy induced modularity (Section 2.6). I also claim that modules (subsystems) of the decomposed monolithic system should not be given unnecessary, surplus information, because it makes them more complicated and larger what is clearly visible in the context of VC-dimension (Section 4.7). Since the functional serial decomposition (Chapter 6) satisfies the above requirements, I have examined its application to neural networks (Chapter 7).

Chapter 2

Entropy based training

Entropy is a key concept of information theory. It measures what is the degree of uncertainty in the random variable. In this chapter I will introduce the notions of entropy and entropy-related measures of information.

2.1 Fundamental concepts

I will begin with basic statistical definitions and properties.

Definition 1 *Two random variables X and Y are* independent *if their distributions satisfy the following equation:*

$$p(X,Y) = p(X) \cdot p(Y),$$

where $p(X,Y)$ denotes joint distribution of X and Y, $p(X)$ and $p(Y)$ — marginal distributions.

We often deal with dependent variables. How should this dependence be measured? It is often suggested covariance can be used as such tool. We start with the definition of the expected value of a random variable.

Definition 2 *The* expected value $\mathrm{E}\{X\}$ *of a random variable X is given by the following equation:*

$$\mathrm{E}\{X\} = \sum_{x \in \mathrm{supp}(X)} x \cdot p(X = x)$$

on condition that $\mathrm{E}\{X\} < \infty$*, where* $\mathrm{supp}(X)$ *is a support of a random distribution* $\mathrm{P}(X)$*.*

The covariance of two random variables X and Y is defined as follows:

Definition 3 *The* covariance $\mathrm{cov}(X,Y)$ *of two random variables X and Y is given by the following equation:*

$$\mathrm{cov}(X,Y) = \mathrm{E}\Big\{(X-\mathrm{E}\{X\})(Y-\mathrm{E}\{Y\})\Big\}.$$

According to the definitions 1, 3 and properties of the expected value, it turns out, that if random variables are independent then $\mathrm{cov}(X,Y)=0$:

$$\begin{aligned}\mathrm{E}\{XY\} &= \sum_{x,y} xyp(X=x,Y=y) = \sum_{x,y} xyp(X=x)p(Y=y) = \\ &= \sum_{x} xp(X=x)\sum_{y} yp(Y=y) = \mathrm{E}\{X\}\mathrm{E}\{Y\} \\ \mathrm{cov}(X,Y) &= \mathrm{E}\{XY\} - \mathrm{E}\{X\}\mathrm{E}\{Y\} - \mathrm{E}\{X\}\mathrm{E}\{Y\} + \mathrm{E}\{X\}\mathrm{E}\{Y\} = \\ &= \mathrm{E}\{XY\} - \mathrm{E}\{X\}\mathrm{E}\{Y\} = 0\end{aligned}$$

However, when $\mathrm{cov}(X,Y)=0$, random variables are not necessarily independent.

Example 1 *Suppose we are given two dependent continuous random variables X and Y, such that $Y=|X|$ and that $\forall_{x\in<-1,1>} p(X=x)=0.5$ and $p(X=x)=0$ otherwise, where $p(X=x)$ denotes probability density function.* $\mathrm{cov}(X,Y) = \int_{-1}^{0} x(-x-0.5)\mathrm{d}x + \int_{0}^{1} x(x-0.5)\mathrm{d}x = 0$.

Since covariance may equal 0 even in the case of obvious and "strong" dependence (Example 1), it is not a good measure of dependency. Although it is used in specific situations (when appropriate assumptions are taken), it does not prove useful in general.

2.2 Entropy

The concept of entropy was first introduced in thermodynamics. Boltzmann noticed that the thermodynamic entropy is related to orderliness and disorderliness in the system. Whereas, the mathematical information theory deals with the syntactic aspect of the transmission of information across a channel. However, these two aspects are closely related.

We would like to develop a useful measure of information $\mathrm{IN}(p)$ connected with an event having probability p. We think of the event as the observance of a symbol having probability p of occurring. Thus we will define information in terms of probability. Certainly $\mathrm{IN}(p)$ should have the following properties:

1. It should be independent of the phenomenon and should not take semantic aspects into consideration.

2. $\mathrm{IN}(p) \geq 0$, since information is a non-negative quantity.
3. If two independent events occur, then the information we get is the sum of the two pieces of information associated with separate events: $\mathrm{IN}(p_1 \cdot p_2) = \mathrm{IN}(p_1) \cdot \mathrm{IN}(p_2)$.
4. Information measure should be a continuous function (functional), of the probability (density). Slight changes in probability distribution should result in slight changes of information.

From the above properties we can derive that:

1. $\mathrm{IN}(p^2) = 2 \cdot \mathrm{IN}(p)$, and further (by induction) $\mathrm{IN}(p^n) = n \cdot \mathrm{IN}(p)$
2. $\mathrm{IN}(p) = \mathrm{IN}\left(\left(p^{1/m}\right)^m\right) = m \cdot \mathrm{IN}\left(p^{1/m}\right) \Longrightarrow \mathrm{IN}\left(p^{1/m}\right) = \frac{1}{m} \cdot \mathrm{IN}(p) \Longrightarrow \mathrm{IN}\left(p^{n/m}\right) = \frac{n}{m} \cdot \mathrm{IN}(p)$
3. From point 2, for $0 < p \leq 1$ and $a > 0$, by continuity, we get: $\mathrm{IN}(p^a) = a \cdot \mathrm{IN}(p)$, which is a property of $\mathrm{IN}(p) = k \cdot \log(p^a)$.

Since $\mathrm{IN}(p)$ is non-negative, $k < 0$. k determines the basis of the logarithm and usually $k = -1/\log(2)$ and $\mathrm{IN}(p) = -\log_2(p)$. (If not otherwise specified, the basis of logarithms in this work is 2).

Suppose now that we have an information channel (random variable) on the alphabet Σ of n symbols $\{a_1, a_2, \ldots, a_n\}$ with corresponding probabilities $\{p_1, p_2, \ldots, p_n\}$. The information associated with the observation of randomly occurring symbol is given by the expected value of IN:

$$\mathrm{H} = \sum_{i=1}^{n} -p_i \cdot \log(p_i)$$

This brings us to a fundamental definition. This definition is essentially due to Shannon in 1948 [74].

Definition 4 Entropy *is a measure of uncertainty of a random variable. The entropy* $\mathrm{H}(X)$ *of a discrete random variable* X *is defined by:*

$$\mathrm{H}(X) = \sum_{x \in \Sigma}^{n} -p(x) \log(p(x)) = \mathrm{E}\left\{\log\left(\frac{1}{p(x)}\right)\right\},$$

where $\mathrm{E}\{.\}$ *denotes the expectation operator. Further,* X *is a discrete random variable with alphabet* Σ *and the probability function:* $p(x) = \mathrm{P_r}(X = x)$, $x \in \Sigma$.

If we have a continuous probability distribution, the obvious generalization of the above definition is as follows:

Definition 5 *In reference to a continuous random variable X described by a probability density function (pdf), f(x), the definition of entropy can be extended as follows:*

$$\mathrm{H_C}(X) = \int_{x\in S} -f(x)\log(f(x))\mathrm{d}x,$$

where S is a domain which includes x. If the pdf $f(x)$ is Rieman integrable in S, discretizing the continuous random variable X into discrete variable X_Δ leads to:

$$\mathrm{H_C}(X_\Delta) + \log(\Delta) \longrightarrow \mathrm{H_C}(X) \quad \text{as} \quad \Delta \longrightarrow 0.$$

To get some feeling for how the Shannon entropy behaves, we now examine one of its most important properties.

Theorem 1 *The maximum of the entropy function of a discrete random variable is the* log *of the number of possible events, and occurs when all the events are equally likely.*

Proof (1) Note that $[\ln(x)]' = 1/x$. From this we find that the tangent to $\ln(x)$ at $x = 1$ is the line $y = x - 1$. Further, since $\ln(x)$ is concave down, we have for $x > 0$, that $\ln(x) \leq x - 1$, with equality only when $x = 1$.
(2) Now given two probability distributions $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_n\}$, where $p_i, q_i \geq 0$ and $\sum_i p_i = \sum_i q_i = 1$, and according to (1), we have the Gibbs inequality:

$$\sum_{i=1}^{n} p_i \ln\left(\frac{q_i}{p_i}\right) \leq \sum_{i=1}^{n} p_i \left(\frac{q_i}{p_i} - 1\right) = \sum_{i=1}^{n}(q_i - p_i) = \sum_{i=1}^{n}(q_i) - \sum_{i=1}^{n}(p_i) = 1 - 1 = 0,$$

with equality only when $\forall_i p_i = q_i$. Since $\log_b(a) = \frac{1}{\log(b)} \cdot \log(a) = c \cdot \log(a)$, the above inequality holds for any base, not just e.
(3) Now we can use the Gibbs inequality to find the probability distribution $P = \{p_1, p_2, \ldots, p_n\}$, $\sum_i p_i = 1$, which maximizes the entropy function. We have

$$\begin{aligned}\mathrm{H}(P) - \log(n) &= \sum_{i=1}^{n} p_i \log(1/p_i) - \log(n) = \sum_{i=1}^{n} \Big(p_i \log(1/p_i) - p_i \log(n)\Big) \\ &= \sum_{i=1}^{n} p_i \Big(\log(1/p_i) - \log(n)\Big) = \sum_{i=1}^{n} p_i \log\left(\frac{1/n}{p_i}\right) \leq 0,\end{aligned}$$

with equality only when $\forall_i p_i = \frac{1}{n}$.

□

It is potent to recognize, that the definitions of information and entropy depend only on the probability distribution. So it is important what model of the data stream one has. It is possible, that two different observers associate different probability distributions to the source, and then they will assign different values to the entropy associated with the source.

It accords with our intuition. Let us consider a university lecture as an example. Two students listening to the same lecture can get two different pieces of information. It is related to their educational background. A good student understands much and for him the words of the lecturer are associated with non-equal probabilities. Whereas, a student who does not understand anything at all, associates equal probabilities to the professor's words and he gets more information, as the entropy for his model is higher.

2.3 Entropy related measures

Now we are ready to introduce entropy related measures. We will begin with the joint entropy of two random variables.

Definition 6 Joint entropy *in reference to a pair of random variables X, Y over the discrete sets χ and η, respectively, is defined as:*

$$\mathrm{H}(X,Y) = -\sum_{x\in\chi}\sum_{y\in\eta} p(x,y)\log(x,y) = \mathrm{E}\Big\{\log(1/p(x,y))\Big\},$$

where $p(x,y)$ denotes the joint probability of occur X and Y.

According to Definition 6, joint entropy is a measure of entire information associated with sources X and Y. Thus, it is symmetric: $\mathrm{H}(X,Y) = \mathrm{H}(Y,X)$.

Definition 7 Conditional entropy *in reference to a pair of random variables X, Y over the discrete sets χ and η, respectively, is defined as:*

$$\begin{aligned}\mathrm{H}(Y|X) &= \sum_{x\in\chi} p(x)\mathrm{H}(Y|X=x)\\ &= -\sum_{x\in\chi} p(x)\sum_{y\in\eta} p(y|x)\log(p(y|x))\\ &= -\sum_{x\in\chi}\sum_{y\in\eta} p(x,y)\log(p(y|x)).\end{aligned}$$

The conditional entropy represents the average of the *degree of uncertainty* of Y over all concrete outcomes of X. In other words, it measures the information that is present in the source Y and is not given by X.

And now we will give the definition of the relative entropy.

Definition 8 Relative entropy $\mathrm{D}(p, q)$ *of the probability mass function* $p(x)$ *with respect to the probability mass function* $q(x)$*, over the set* χ*, is defined by:*

$$\mathrm{D}(p||q) = \sum_{x \in \chi} p(x) \log\left(\frac{p(x)}{q(x)}\right).$$

It is not immediately obvious what is the use of the relative entropy, or why it should be a good measure of a distance between the two distributions. The following theorem gives the reason why it is regarded as a distance measure.

Theorem 2 *The relative entropy is non-negative,* $\mathrm{D}(p, q) \geq 0$*, with equality if and only if* $p(x) = q(x)$ *for all* x*.*

Proof See step (2) of the proof of the Theorem 1.

The relative entropy is very useful, not in itself, but because other entropic quantities can be regarded as special cases of the relative entropy. The following very useful definition is a good example.

Definition 9 *The* mutual information *between two random variables* X *and* Y *is defined as:*

$$\mathrm{I}(X : Y) = \sum_{x \in \chi} \sum_{y \in \eta} p(x, y) \log\left(\frac{p(x, y)}{p(x)p(y)}\right) = \mathrm{D}\Big(p(x, y)||p(x)p(y)\Big),$$

where D *represents the relative entropy.*

Taking the definition of relative entropy into consideration, it becomes clear, that *mutual information* is a measure of information, which the two sources X and Y have in common. It offers a correlation between X and Y and it is for sure a better measure than covariance (see Example 1 form Section 2.1. Obviously, mutual information is always symmetric: $\mathrm{I}(X : Y) = \mathrm{I}(Y : X)$.

The relations between entropy, conditional entropy and mutual information may mostly be deduced from the 'entropy Venn diagram' shown in Figure 2.1. Such figures are not completely reliable as a guide to the properties of entropy, but they provide a useful mnemonic for remembering the various definitions and properties.

For the sake of clarity will present several properties of entropy and entropy related measures.

Theorem 3 *Properties of entropy:*

1. $\mathrm{H}(X) \geq 0$

2. $\mathrm{I}(X : Y) \geq 0$*, with equality when* X *and* Y *are independent.*

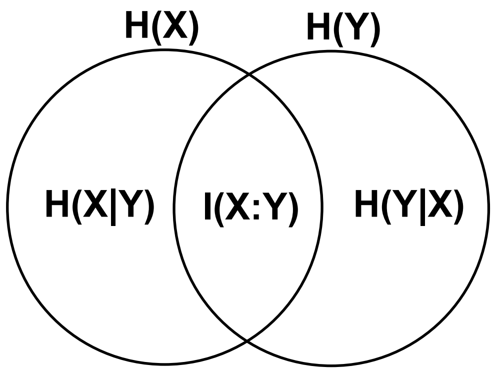

Figure 2.1: Relations between different entropies.

3. $\mathrm{H}(X|Y) = \mathrm{H}(X,Y) - \mathrm{H}(Y)$.
4. $\mathrm{I}(X:Y) = \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(X,Y) = \mathrm{H}(X) - \mathrm{H}(X|Y)$
5. $\mathrm{H}(X,Y) \leq \mathrm{H}(X) + \mathrm{H}(Y)$ *with equality if and only if X and Y are independent random variables.*
6. *Conditioning reduces entropy: For any two random variables X and Y* $\mathrm{H}(Y|X) \leq \mathrm{H}(Y)$ *with equality in each if and only if X and Y are independent random variables.*
7. $\mathrm{H}(X_1, X_2, \ldots, X_n) \leq \sum_{i=1}^{n} \mathrm{H}(X_i)$, *with equality if and only if the random variables X_i are independent.*
8. $\mathrm{H}(X) \leq \log|\chi|$ *with equality if and only if X is uniformly distributed over χ.*
9. *If* $\mathrm{H}(X) = 0$, *the variable X describes deterministic process and there is an absolute certainty that only one outcome of X is possible.*

Proof

1. Obvious from Definition 4, since we have $0 < p(x) < 1$ and $\log(p(x)) < 0$.
2. We again use the fact that $-\log(x) \geq (1-x)/\ln(2)$ for all positive x, with equality if and only if $x = 1$. We find that

$$\begin{aligned}
\mathrm{I}(X:Y) &= \sum_{x,y} -p(x,y)\log\left(\frac{p(x)p(y)}{p(x,y)}\right) \\
&\geq \frac{1}{\ln(2)} \sum_{x,y} p(x,y)\left(1 - \frac{p(x)p(y)}{p(x,y)}\right) \\
&= \frac{1}{\ln(2)} \sum_{x,y} p(x,y) - p(x)p(y) = \frac{1-1}{\ln(2)} = 0.
\end{aligned}$$

Note, that when X and Y are independent, $p(x,y) = p(x)p(y)$, and $\mathrm{I}(X:Y) = \sum_{x,y} p(x,y)\log(1) = 0$.

3. From the relevant definitions we get:

$$\begin{aligned}\mathrm{H}(X|Y) &= \sum_y p(y) \sum_x -\frac{p(x,y)}{p(y)} \log\left(\frac{p(x,y)}{p(y)}\right) \\ &= \sum_y \sum_x -p(x,y)\log(p(x,y)) - \sum_y \sum_x -p(x,y)\log(y) \\ &= \mathrm{H}(X,Y) - \mathrm{H}(Y).\end{aligned}$$

4. In the same way as above, using the relevant definitions.
5. Follows from points 2. and 4.
6. Follows from point 4.
7. Follows from point 5. by induction.
8. Follows from Theorem 1.
9. Follows from Definition 4 and the fact that $\lim_{x\to 0} x\log(x) = 0$.

□

So far, I have only given the definitions and entropy related properties necessary for the remaining part of the work. More details concerning the information theory can be found for example in [10, 51, 52, 74].

2.4 Learning and information theory

Information theory has been developed and used to describe the processes of communication. In particular original works by Shannon characterized the capacity of information channel [74]. It turns out that the training process of an information system also can be described in terms of entropy and entropy related measures [21, 48]. Suppose we are given an unknown system and the objective is to train the model of this system, so that the behaviour of the model resembles the original system as much as it is possible. In the language of information theory it means that we want to create an information channel that models the given one. Figure 2.2 depicts this situation.

Let us assume that we have a communication process with an input X and output Y, where X and Y are random variables. The *transmission rate* of such channel is the amount of information that random variables X and Y have in common and is

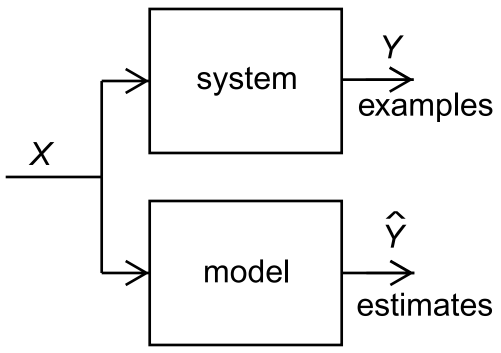

Figure 2.2: Learning described in the language of communication channels.

depicted by the mutual information $\mathrm{I}(X : Y)$. The transmission takes place when $\mathrm{I}(X : Y) > 0$. The better the communication channel, the larger $\mathrm{I}(X : Y)$. The *cannel capacity* defines the upper-bound on on the rate of transmission as a function of all possible input distributions.

In case of learning we have an a priori unknown system, considered as a communication process with input X and output Y, to be modelled by a trained neural network, also considered as a communication process with the same input X and output $\hat{Y}$. In case of a well-trained model, the output $\hat{Y}$ will approximate the output Y.

Like communication processes, which are bounded by the channel capacity, learning processes are also bounded by a *learning bound*. The behaviour of the unknown system must be learned from a sequence of observed input-output pairs (x, y), belonging to the alphabet (the range) of the corresponding random variables X and Y. These pairs are called the training set. If random variables X and Y are independent, there is nothing to be learned. The behaviour of the random variable Y given X can excellently be modelled if Y and X are strongly dependent. This means that the mutual information is high. It happens when the input information, like in communication channel, is mostly propagated towards the output of the system. Now we can measure the amount of information that can be learned by the model. Like in the communication channel, it is described by the mutual information $\mathrm{I}(X : Y)$ between X and Y. According to Definition 9, this amount of information depends on the distribution of the input samples. If this distribution is properly chosen, the mutual information reaches a maximum: the *learning bound*.

Now we can apply the already defined entropy-related measures. The progress of learning can be measured by the uncertainty about the observed output Y given the estimated output $\hat{Y}$. This uncertainty is given by the conditional entropy $\mathrm{H}(Y|\hat{Y})$, which is certainly bounded by $0 \leq \mathrm{H}(Y|\hat{Y}) \leq \mathrm{H}(Y)$ (see Theorem 3). In the case of successful learning, the model channel approximates the system channel and consequently the conditional entropy approaches zero. It is in the case, when $Y = f(\hat{Y})$, where $f(.)$ is a certain function. Since the entropy does not take into consideration any semantic aspects of information, even if the learning has finished successfully, there is a need to transform (code) the output of the model, so that the alphabets of $\hat{Y}$ and Y are corresponding.

If the uncertainty is not diminished by the learning process and nothing has been learned, certainly Y and $\hat{Y}$ are independent, and $\mathrm{H}(Y|\hat{Y}) = \mathrm{H}(Y)$. The amount of learned information can be computed by $\mathrm{H}(Y) - \mathrm{H}(Y|\hat{Y})$, which, according to Theorem 3, equals $\mathrm{I}(Y : \hat{Y})$:

$$\mathrm{I}(Y : \hat{Y}) = \mathrm{H}(Y) - \mathrm{H}(Y|\hat{Y}).$$

It is important to emphasize the essential difference between traditional (distance-related) measure of similarity $Y - \hat{Y}$ and the mutual information: $\mathrm{I}(Y : \hat{Y})$. Even if the training set is neither numeric (1, 2, 3. . .) nor ordered (a, b, c. . .), for example feelings (angry, peaceful, hungry. . .), mutual information can still be used to measure the achieved learning.

2.5 Deterministic model training

Till now we have not made any assumptions about the model. In modelling we generally want the model to be deterministic. In such case there is no uncertainty about $\hat{Y}$ given X as $\hat{Y}$ is a function of X:

$$\mathrm{H}(\hat{Y}|X) = 0.$$

This conditional entropy can be written as (Theorem 3) $\mathrm{H}(\hat{Y}|X) = \mathrm{H}(X, \hat{Y}) - \mathrm{H}(X)$ and leads to the conclusion that:

$$\mathrm{H}(X, \hat{Y}) = \mathrm{H}(X).$$

Adding a condition to the entropy reduces the entropy. Since the entropies are positive values we have:

$$\mathrm{H}(\hat{Y}|(X, Y)) = \mathrm{H}(X, Y, \hat{Y}) - \mathrm{H}(X, Y) = 0.$$

Now we are ready to prove the following theorem:

Theorem 4 *Data processing leads to erosion of data.*

$$\mathrm{H}(Y|\hat{Y}) \geq \mathrm{H}(Y|X).$$

The uncertainty about Y is less using the original data X instead of the processed data $\hat{Y}$. Equality holds if there is no erosion. That means that there is a bijective relation between X and $\hat{Y}$.

Proof $\mathrm{H}(Y|\hat{Y}) \geq \mathrm{H}(Y|(X, \hat{Y})) = \mathrm{H}(X, Y, \hat{Y}) - \mathrm{H}(X, \hat{Y}) = \mathrm{H}(X, Y) - \mathrm{H}(X) = \mathrm{H}(Y|X)$.

□

From the above-theorem directly flows that the amount of information that can be learned is bounded by the *data rate* $\mathrm{I}(X : Y)$. We will give this inequality in the form of the theorem:

Theorem 5 *The amount of learned information* $\mathrm{I}(Y : \hat{Y})$ *is bounded by the data rate of the system* $\mathrm{I}(X : Y)$*:*

$$\mathrm{I}(Y : \hat{Y}) \leq \mathrm{I}(X : Y).$$

It articulates the obvious truth, that only available knowledge can be extracted.

However, there the problem of the training samples remains. The training set can be prepared in different ways. The upper bound of learning in Theorem 5 strongly depends on the distribution of the training set. In other words, it is important which samples have been chosen to train the model. For example, if the model is trained with identical samples, the data rate is zero, and nothing can be learned. Whereas, in the case of properly adapted training set, the results can be satisfying. Since the data rate for every existing channel (as well as for an existing information system) is limited, there exists such distribution of input data X which maximizes $\mathrm{I}(X : Y)$.

2.6 Modularity

Monolithic learning has been extensively studied so far, but without spectacular success. In the case of neural networks it is mainly caused by the inability to handle data interference. For large structures the time of training is excessively long and the results are poor. The remedy for this situation is modularity. Unfortunately, modular neural networks lack a constructive template. The desired outcome is usually hard to achieve.

It occurs, that the entropy can indicate how to structure and decompose the trained system. Thanks to entropy-based measures we can identify the independent parts of the model, which can be learned separately and then joined together. This is a novel and promising approach, since it can be applied generally to any information system.

Suppose we are given an information system with several inputs and outputs described respectively by multidimensional random variables $X = (X_1, X_2, \ldots, X_n)$ and $Y = (Y_1, Y_2, \ldots, Y_n)$, as it is depicted in Figure 2.3.

2.6.1 Argument reduction

Let $\Pi_1 = (X_i, X_j, \ldots)$ and $\Pi_2 = (X_k, X_l, \ldots)$ be a disjoint partitions of the input X (Figure 2.4), such that $\Pi_1 \cap \Pi_2 = \emptyset$ and $\Pi_1 \cup \Pi_2 = X$. If the partitioning of input variables can be done in such a way that $\mathrm{I}(Y : X) = \mathrm{I}(Y : \Pi_1)$, the partition Π_2 can be removed from the input. It should be emphasized that the mutual information is calculated between the reduced input of the model and the output of the system. In other words, the information necessary to compute Y is given by the partition Π_1 and the information supported by Π_2 is either redundant or useless and can be eliminated without loss. In this case, since $\mathrm{H}(X) \geq \mathrm{H}(\Pi_1)$, the entropy of input can be reduced and

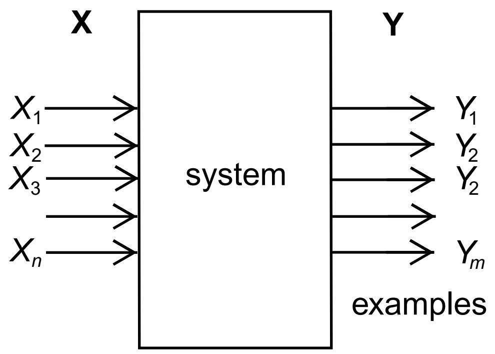

Figure 2.3: System of several inputs and outputs.

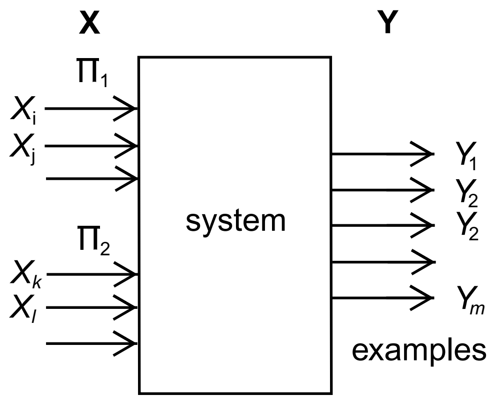

Figure 2.4: The partition of the inputs.

the complexity of the model can by diminished, because it has to deal with the reduced amount of information (Figure 2.5).

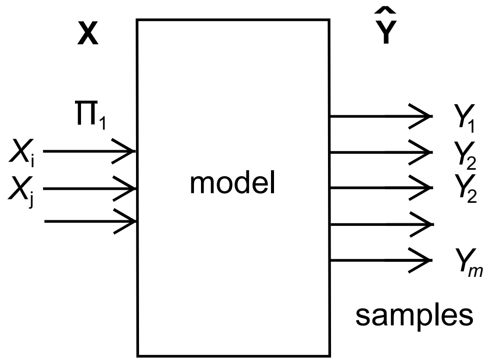

Figure 2.5: The argument reduction of the model.

The algorithms suitable for finding the desired partitioning will be discussed later.

2.6.2 Serial decomposition

Sometimes it is impossible to find the partitioning described in section 2.6.1 and the argument reduction cannot be directly applied. However, in this situation, the functional serial decomposition proves useful.

Again, let $\Pi_1 = (X_i, X_j, \ldots)$ and $\Pi_2 = (X_k, X_l, \ldots)$ be a disjoint partitions of the input X (Figure 2.4), such that $\Pi_1 \cap \Pi_2 = \emptyset$ and $\Pi_1 \cup \Pi_2 = X$. The model of the system can be designed in a serial way as it is depicted in Figure 2.6.

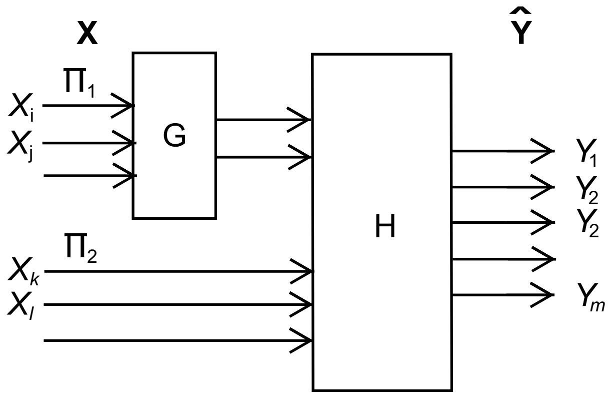

Figure 2.6: The serial decomposition of the model.

If we are able to design a function G with the input Π_1 in a way that $\mathrm{I}(X : Y) = \mathrm{I}(G(\Pi_1), \Pi_2 : Y)$, the model can be designed in modular way and the two blocks G and H can be trained separately. It is important to stress that no information is lost. The optimal performance of the trained model is the same as without decomposition. However, the process of training can be faster and more efficient. This is a very beneficial situation, because we not only achieve the modularity but also clear indications how to train the separate blocks. The key issue for every system is to find a proper partitioning and the function G. Fortunately, the algorithms suitable for this task have been deeply studied so far and proved very useful in digital circuits design [40, 41, 42, 57, 71, 72]. They will be addressed intensely in the later part of this work.

The above approach can be easily extended to situation, when the input X is divided into many disjoint partitions $\Pi_1, \Pi_2, \ldots$ with corresponding blocks H_i.

2.6.3 Parallel decomposition

So far we have not tried to partition the output of the system. It turns out, that the notion of argument reduction, introduced in Section 2.6.1, can be easily adapted to the process named parallel decomposition. Let us consider again the system in Figure 2.3. Let $\Gamma_1 = (Y_i, Y_j, \ldots)$ and $\Gamma_2 = (Y_k, Y_l, \ldots)$ denote the disjoint partition of Y, such that $\Gamma_1 \cap \Gamma_2 = \emptyset$ and $\Gamma_1 \cup \Gamma_2 = Y$. The model of the system consists of two blocks G_1, G_2 working in parallel and the output of the model consists of the merged outputs of blocks G_1 and G_2: $\hat{Y} =< G_1(X_1), G_2(X_2)) >$. Figure 2.7 describes the above situation. Note, that $X_1 \subseteq X$ and $X_2 \subseteq X$ are not necessarily disjoint. They are such subsets of X that hold only indispensable information to compute Γ_1 and Γ_2. In the entropy terms it is expressed in the following way, respectively: $\mathrm{I}(\Gamma_1 : X_1) = \mathrm{I}(\Gamma_1 : X)$ and $\mathrm{I}(\Gamma_2 : X_2) = \mathrm{I}(\Gamma_2 : X)$.

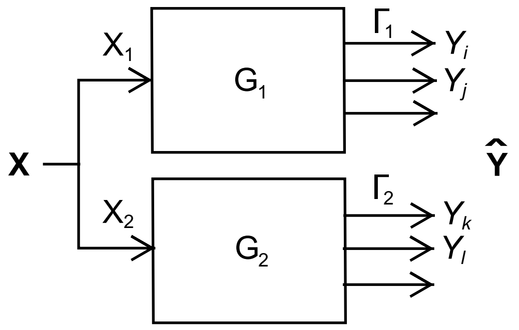

Figure 2.7: The parallel decomposition of the model.

The application of the already described argument reduction to the inputs of G_1 and G_2 results in the fact that X_1 and X_2 are the minimal partitions of X suitable for computing Γ_1 and Γ_2. As the unnecessary information is removed, we obtain that $\mathrm{H}(X_1) \leq \mathrm{H}(X)$ and $\mathrm{H}(X_2) \leq \mathrm{H}(X)$, where H denotes entropy. The blocks of such model are smaller than in the case of monolithic design and can be easily trained separately.

Certainly, the output can be divided into more than two disjoint partitions, each with the corresponding block G. Such approach is often used in machine learning. Suppose we are to design the system capable of recognizing the characters. The monolithic solution is to use one big neural network with 26 outputs, each corresponding to exactly one letter of the alphabet. However, it turns out that better results are achieved when 26 smaller separate neural networks are used.

The algorithmic aspect of parallel decomposition is the same as for the problem of argument reduction.

Chapter 3

Multilayer neural networks

Multilayer feedforward network structures are characterized by directed weighted layered graphs. The vertices of this graph represent simple computing units — called *neurons* and the directed edges describe the connections between neurons. Each neuron has one output and many inputs from the neurons of preceding layer. The first layer is called *input layer* and the last layer is an *output layer*. The remaining layers are denoted as *hidden layers*.

The *architecture* of the neural network is a set of directed weighted layered graphs which differ only in the values of edges. An example two hiddenlayer neural network is depicted in Figure 3.1.

3.1 Neural network fundamentals

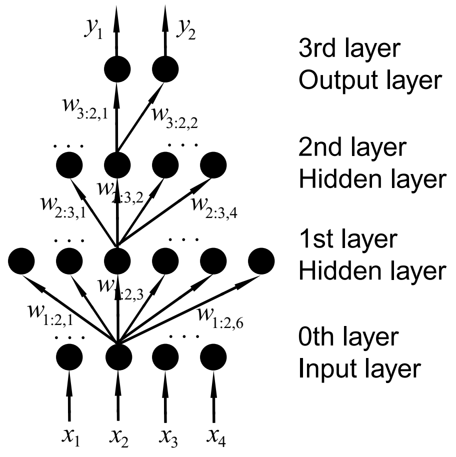

Figure 3.1: A two hidden layer neural network.

The computation in feedforward neural networks is done in the neurons. Each neuron is stimulated by its weighted inputs. The output of the neural unit is a nonlinear function (called *activation function*) of weighted sum of its inputs from the preceding layer:

$$a_{i:j} = F_{i:j}(c_{i:j}) = F_{i:j}\left(\sum_{k=1}^{s_{i-1}} w_{i:j,k} \cdot a_{i-1:k} + b_{i:j}\right),$$

$$c_{i:j} = \sum_{k=1}^{s_{i-1}} w_{i:j,k} \cdot a_{i-1:k} + b_{i:j},$$

where i denotes the layer and j is the neuron number. $a_{i:j}$ is the output of jth neuron in ith layer, $w_{i:j,k}$ is a weight of the link from kth neuron in layer $(i-1)$ to the jth neuron in ith layer, $c_{i:j}$ is the excitation of jth neuron in ith layer, the function $F_{i:j}(.)$ is a nonlinear activation function of $c_{i:j}$, $b_{i:j}$ is the bias for jth neuron in layer i, s_i indicates the number of neurons in layer i. The last layer is the output layer and the 0th layer is the input layer.

The non-linear functions $F_{i:j}(.)$ are the heart of the neural network. If these functions were replaced by linear ones, neural network, since the superposition of linear functions is also a linear function, would be nothing more than just a linear function. Generally in feedforward neural networks S-shaped functions are usually used as $F_{i:j}(.)$. It is caused by the fact that they are continuous, monotonic, bounded and they have a positive derivative. These properties are very important in gradient descent methods of training. Among very popular sigmoid functions are: logistic function $f(x) = \frac{1}{1+e^{(-x)}}$ (Figure 3.2) and hyperbolic tangent $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ (Figure 3.3).

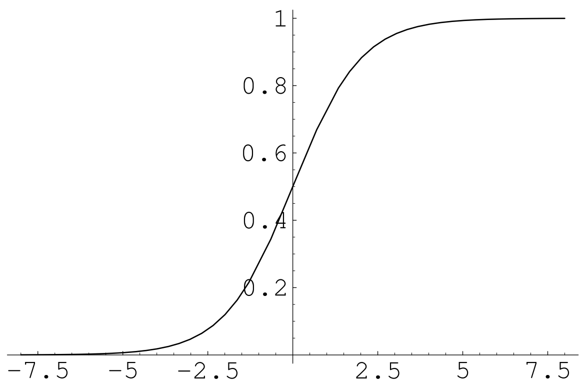

Figure 3.2: The logistic function.

The key issue in neural network training for a given architecture is assigning such weights to interneuronal connections so that the error of the output of the entire network is minimized. The training algorithms will be discussed later.

What are the capabilities of the above introduced structure and what the architecture should be like? In the next section we will address this question in the context of the 23 problems of Hilbert.

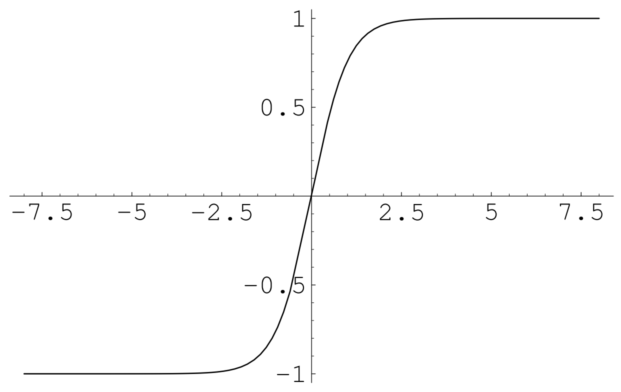

Figure 3.3: The hyperbolic tangent.

A number of approaches that demonstrate the power of neural networks have been published so far. Almost all of them rely on some previously established method of representing or approximating a function $f(x)$ through an expansion in terms of a given training set of basis functions. In the later section we will follow the Kolmogorov solution and Cybenko results.

3.2 Exact and approximate representations

The synthesis of a continuous real-valued function on the cube $I^d = [a,b]$ in $\mathbf{R}^d$, using a neural network-like representation admits an exact solution as a consequence of Kolmogorov's solution to Hilbert's 13th problem. We will present the simplified form of this theorem, given by Sprecher [26].

Theorem 6 *Let λ_k be a sequence of positive integrally independent numbers; i.e., $\sum_k r_k \lambda_k \neq 0$ for any integers r_k that are not all 0. Letting $D = [0, 1 + \frac{1}{5!}]$, there exists a continuous monotonically increasing function $\Psi : D \to D$ having the following property: For every real-valued continuous function $f : [0,1]^d \to \mathbf{R}$ with $d \geq 2$ there is a continuous function Φ and constants $\{a_d\}$, $\{b_m\}$ such that:*

$$f(x_1, \ldots, x_d) = \sum_{k=0}^{2d} \Phi \left(b_k + \sum_{j=1}^{d} \lambda_j \Psi(x_j + k a_d) \right).$$

The Kolmogorov solution can be implemented by a net with two hidden layers. The first layer contains $d(2d+1)$ copies of Ψ, and the second layer $2d+1$ copies of the function Φ. The output is achieved by summing all the outputs of the second layer.

The crucial problem with this solution is that the node function Φ in the second hidden layer depends on the choice of function f, and, since it is continuous, it is actually complicated. Thus the synthesis of different functions of several variables requires us to synthesize functions of a single variable. It is much more than just adapt weights $\{\lambda_j\}$, $\{a_d\}$ and $\{b_k\}$. Moreover, these functions could be very non-smooth even for a differentiable function f. Furthermore, Kolmogorov's theorem is only an existance result.

Such approach, since it is not general, is not especially useful. However, its modifications lead to approximate representations of the given function by the neural network.

We will follow Cybenko [20], but first of all we will introduce the necessary notions of the space of functions computable be a neural network. Since here we use underlining (_) to denote vector variables.

Definition 10 *The class $\mathcal{N}_{1,\sigma} = \{\eta_1\}$ of real-valued functions exactly construable by a single hidden layer feedforward network with a single linear output node and hidden layer node nonlinearity $\sigma(.)$ is the linear span of functions of $\underline{x} \in \mathbf{R}^d$ of the form $\sigma(\underline{w} \cdot \underline{x} - \tau)$.*

In other words, $\mathcal{N}_{1,\sigma}$ is the set of functions η_1 of $\underline{x}$, specified by:

$$\eta_1(\underline{x}, \underline{w}) = a_{2:1}(\underline{x}) = \sum_1^{s_1} w_{2:1,i} \cdot \sigma(c_{1:i}) - \tau_{2:1}, \text{where} \tag{3.1}$$

$$c_{1:i}(\underline{x}) = \sum_1^{d} w_{1:i,k} \cdot x_k - \tau_{1:i}. \tag{3.2}$$

Equations 3.1 and 3.2 clearly show, that functions in the set $\mathcal{N}_{1,\sigma}$ are sums of scaled and translated functions $\sigma(.)$. For example, if the node function $\sigma(.)$ is logistic or tanh then the polynomials and trigonometric functions cannot be found in $\mathcal{N}_{1,\sigma}$. If $\sigma(.)$ is a polynomial of degree k then all the functions in $\mathcal{N}_{1,\sigma}$ are also polynomials of degree k.

Let $\mathcal{C}(\mathcal{X})$ is the space of continuous functions on $\mathcal{X}$ and $d(f, g)$ is the worst-case sup-norm metric:

$$d(f, g) = \sup_{x \in \mathcal{X}} |f(x) - g(x)|.$$

A set of functions $\mathcal{F}$ can *arbitrarily close approximate* to a set of functions $\mathcal{G}$, in the sense of a given metric d, if to any $g \in \mathcal{G}$ and to any positive ϵ there is an $f \in \mathcal{F}$ that is at least that close to g. We will state it again in the form of the definition.

Definition 11 *$\mathcal{F}$ is* dense in $\mathcal{G}$ *if*

$$\forall g \in \mathcal{G}\ \forall \epsilon > 0\ \exists f \in \mathcal{F} \quad d(f, g) < \epsilon.$$

When $\mathcal{F}$ is a subset of $\mathcal{G}$, then the functions in $\mathcal{G}$ that can be arbitrarily closely approximated by the functions in $\mathcal{F}$ are the ones in the subset $\mathcal{F}$ together with the *limit points* of this set of functions. The set $\mathcal{F}$ and its limit points is known as the closure $\bar{\mathcal{F}}$ of $\mathcal{F}$ in $\mathcal{G}$ with respect to the metric d, and is specified as follows:

Definition 12 *Given a metric d on a space of functions $\mathcal{G}$ the* closure $\bar{\mathcal{F}}$ *of a family $\mathcal{F} \subset \mathcal{G}$ of functions is:*

$$\bar{\mathcal{F}} = \{g \in \mathcal{G} : \forall \epsilon > 0 \; \exists f \in \mathcal{F} \quad d(g, f) < \epsilon\}.$$

In other words, $\mathcal{F}$ is precisely the set of functions in $\mathcal{G}$ that can by arbitrarily closely approximated by functions in $\mathcal{F}$.

According to the above definition, $\bar{\mathcal{N}}_{1,\sigma}$ is the set of all functions which can be arbitrarily closely approximated by single hidden layer neural networks with a linear output node and hidden layer node function $\sigma(.)$.

Not every function is suitable to play the role of nonlinearity in neural network. It turns out that only functions satisfying the restriction given in the below definition, called *discriminatory functions*, are appropriate for neural processing.

Definition 13 *$\sigma(.)$ is* discriminatory *if for any finite, signed, regular measure μ*

$$\forall \underline{w} \in \mathbf{R}^d \; \forall \tau \in \mathbf{R} \int_{I^d} \sigma(\underline{w} \cdot \underline{x} - \tau) d\mu(\underline{x}) = 0 \tag{3.3}$$

implies that $\mu = 0$.

Definition 14 *A sigmoidal function $\sigma(.)$ is one such that*

$$\infty > \lim_{x \to \infty} \sigma(x) = \bar{\sigma} > \underline{\sigma} = \lim_{x \to -\infty} \sigma(x) > -\infty.$$

implies that $\mu = 0$.

However, it occurs that the most popular logistic function is discriminatory.

Lemma 1 *Any bounded, measurable sigmoidal function $\sigma(.)$ is discriminatory. Thus any bounded, piecewise continuous sigmoidal function (example linear threshold function, logistic, tanh) is discriminatory.*

Proof sketch Any sigmoid function is capable of approximating the step function (Figure 3.4). We have an approximate unit-step function defined as follows:

$$\hat{U}_a(x) = \frac{\sigma(ax) - \underline{\sigma}}{\bar{\sigma} - \underline{\sigma}}.$$

The resemblance can be arbitrarily accurate, since

$$\forall |x_0| > 0 \quad \lim_{a \to \infty} \sup_{x : |x| > |x_0|} |\hat{U}_a(x) - U(x)| = 0.$$

Furthermore, step functions can approximate trigonometric functions. Finally, by the uniqueness of Fourier transform, trigonometric functions are discriminatory.

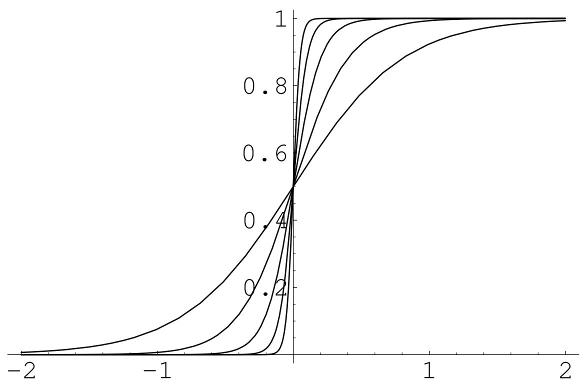

Figure 3.4: A family of the unit-step function approximations.

□

Having identified an important class of discriminatory node functions, we are ready to state the Cybenko theorem about computational abilities of neural networks.

Theorem 7 *If* $\sigma(.)$ *is continuous and discriminatory, then functions* $\mathcal{N}_{1,\sigma}$ *implemented by a single hidden layer net are dense in* $\mathcal{C}(I^d)$*, or equivalently stated,*

$$\bar{\mathcal{N}}_{1,\sigma} = \mathcal{C}(I^d).$$

Proof sketch [20] Suppose $\bar{\mathcal{N}}_{1,\sigma} \neq \mathcal{C}(I^d)$. If the closed, linear subspace $\bar{\mathcal{N}}_{1,\sigma} \neq \mathcal{C}(I^d)$, then, according to Hahn-Banach Theorem [68], there exists a bounded, linear functional L on $\mathcal{C}$ such that

$$\forall f \in \bar{\mathcal{N}}_{1,\sigma} \; L(f) = 0 \quad \text{and} \quad \exists g \in \mathcal{C} \; L(g) \neq 0. \tag{3.4}$$

Using Riesz Representation Theorem [6] L can be given an integral representation:

$$L(f) = \int_{I^d} f(\underline{x})\mu(d\underline{x}),$$

where μ is a finite, signed, regular measure. One can think of the Stieltjes integral more informally:

$$L(f) = \int_{I^d} f(\underline{x})d\mu'(\underline{x})d\underline{x}.$$

Since, $f \in \bar{\mathcal{N}}_{1,\sigma}$ is a weighted sum of translated functions $\sigma(.)$

$$f(x) = \sum_{i=1}^{n} \alpha_i \sigma(\underline{w}_i \cdot x - \tau_i).$$

From the linearity of the functional L and Equation 3.4 arises that

$$L(f) = L\left(\sum_{i=1}^{n} \alpha_i \sigma(\underline{w}_i \cdot x - \tau_i)\right) = \sum_{i=1}^{n} \alpha_i L(\sigma(\underline{w}_i \cdot x - \tau_i)) = 0.$$

Because this must hold for all choices of parameters, it holds if and only if

$$\forall \underline{w}, \tau \quad L\left(\sigma(\underline{w}_i \cdot x - \tau_i)\right) = 0.$$

The fact that $\sigma(.)$ is continuous and discriminatory implies that $\mu = 0$ and we conclude that L must be identically zero in contradiction Equation 3.4. Hence, the desired contradiction is reached and $\bar{\mathcal{N}}_{1,\sigma} = \mathcal{C}(I^d)$.

$\square$

In other words, a single hidden layer neural network with sigmoid activation function can approximate arbitrarily closely any continuous function of $\underline{x}$.

3.3 Neural network training

Suppose we are given a training set of n input-output pairs

$$\mathcal{T} = \{(\underline{x}_m, \underline{t}_m), m = 1 \ldots n\}$$

and we wish to select a neural network η so that the output $\underline{y}_i = \eta(\underline{x}_i, \underline{w})$ is "close" to the desired output $\underline{t}_i$ for input $\underline{x}_i$. Usually this closeness is formalized through an error of the form

$$\mathcal{E}_{\mathcal{T}} = \frac{1}{2} \sum_{m=1}^{n} \|\underline{y}_m - \underline{t}_m\|^2.$$

Certainly, the error $\mathcal{E}_{\mathcal{T}} = \mathcal{E}_{\mathcal{T}}(\underline{w})$ is a function of the vector of weights of the neural network $\underline{w}$.

We are thus confronted with a nonlinear optimization problem:

$$\min_{\underline{w} \in \mathcal{W} \subset \mathbf{R}^p} \quad \mathcal{E}_{\mathcal{T}}(\underline{w}).$$

The difficulty of the above problem is caused by the fact, that usually the dimension of the weight space $\mathcal{W}$ is very high. Networks with hundreds or thousands of weights are often encountered in for example image processing applications. It excludes direct search methods and analytical solutions. The second inherent difficulty is that the function $\mathcal{E}_{\mathcal{T}}(\underline{w})$ is highly irregular, with many local minima and large regions of little slope (for example in the case of saturation of $\sigma(.)$ nonlinear function). This is mainly caused by the fact that the neural network approximation of the given function is not unique (for example the networks with permutations of nodes in layers are indistinguishable).

The most popular training methods are those basing on the gradient of the error function. Introduce the *gradient vector*, the vector of first partial derivatives,

$$\underline{g}(\underline{w}) = \nabla \mathcal{E}_{\mathcal{T}}|_{\underline{w}} = \left[\frac{\partial \mathcal{E}_{\mathcal{T}}}{\partial w_i}\right]\Bigg|_{\underline{w}}.$$

The negative of gradient vector points in the direction of steepest decrease of $\mathcal{E}_{\mathcal{T}}$. Also introduce the *Hessian matrix* of the second partial derivatives:

$$H(\underline{w}) = [H_{i,j}(\underline{w})] = \nabla^2 \mathcal{E}_{\mathcal{T}}(\underline{w}),$$

$$H_{i,j} = \frac{\partial^2 \mathcal{E}_{\mathcal{T}}(\underline{w})}{\partial w_i \partial w_j} = H_{j,i}.$$

The Taylor's series for error function $\mathcal{E}_{\mathcal{T}}(\underline{w})$ can now be approximated as

$$\mathcal{E}_{\mathcal{T}}(\underline{w}) \approx m(\underline{w}) = \mathcal{E}_{\mathcal{T}}(\underline{w}^0) + \underline{g}(\underline{w}^0)^T(\underline{w} - \underline{w}^0) + \frac{1}{2}(\underline{w} - \underline{w}^0)^T H(\underline{w}^0)(\underline{w} - \underline{w}^0), \tag{3.5}$$

where $m(\underline{w})$ denotes a quadric model of function $\mathcal{E}_{\mathcal{T}}(\underline{w})$ at point $\underline{w}$.

If the Hessian is a *positive definite matrix* we can easily find the minimum of the model $m(\underline{w})$ given by the Equation 3.5. We begin with determining the stationary point. First of all we calculate the gradient of $m(\underline{w})$.

$$\left[\frac{\partial m}{\partial w_i}\right]\Bigg|_{\underline{w}} = \nabla m(\underline{w}) = \underline{g}(\underline{w}^0) + H(\underline{w}^0)(\underline{w} - \underline{w}^0). \tag{3.6}$$

The stationary point $\underline{w}^*$ is the one that $\nabla m(\underline{w}^*) = 0$. Since $H(\underline{w}^0)$ is a positive definite matrix, such point also minimizes the model $m(\underline{w})$. It arises from Equation 3.6 that

$$\underline{w}^* = \underline{w}^0 - (H(\underline{w}^0))^{-1}\underline{g}(\underline{w}^0) = \underline{w}^0 - H^{-1}\underline{g}. \tag{3.7}$$

The above equation indicates, assuming quadric model and on condition that H is positive definite, that in the process of learning the weights $\underline{w}$ of the neural network should be updated in the direction

$$\Delta \underline{w} = -H^{-1}\underline{g}, \tag{3.8}$$

$$\underline{w}_{k+1} = \underline{w}_k + \Delta \underline{w}_k, \tag{3.9}$$

where the Hessian matrix H and the gradient $\underline{g}$ of $\mathcal{E}_{\mathcal{T}}(\underline{w})$ are calculated at the point $\underline{w}_k$.

Unfortunately, in the contemporary neural networks the calculation of the Hessian is impossible due to the large (even thousands) number of weights. However, it is replaced with a positive, sometimes adaptive, constant or it is approximately evaluated. Independently of the details, almost all learning methods require us to evaluate $\mathcal{E}_{\mathcal{T}}(\underline{w})$ and $\underline{g}(\underline{\mathrm{w}})$ repeatedly as we iteratively get a better approximation of the training set. Such calculation must be carried out efficiently, because they are repeated many times (millions of times in many cases) in the course of operation of a training algorithm. Fortunately, the widely relied-on process of backpropagation is an effective method of calculating these values.

3.3.1 Gradient evaluation

Further study of backpropagation algorithm in the context of feedforward neural networks requires us to recall and complete the notation already introduced. For the sake of clarity and without loss of generality we follow common practise and consider only a single output neural network.

1. The collection of inputs form a $d \times n$ matrix S whose mth column is $\underline{x}_m = \{x_i^m\}$. The corresponding collection of outputs form a n dimensional vector $\{t_m\}$.
2. Let i denote the ith layer, with the inputs occurring in the 0th layer and with the outputs in the last layer being the Lth.
3. The layer will be indexed as the first subscript and separated from other subscripts by a colon (:).
4. The number of nodes in the ith layer is given by the "width" s_i.
5. The jth node function in layer i is $F_{i:j}$, which is also denoted as $\sigma_{i:j}$ or σ.
6. The argument of $F_{i:j}$, when $\underline{x}_m$ is the input to the net is indicated as $c_{i:j}^m$.
7. The value of $F_{i:j}(c_{i:j}^m) = a_{i:j}^m$. In the case of network input (0th layer) $\{a_{0:j}^m\} = \underline{x}_m$
8. The s_i-dimensional vector $\underline{a}_i^m$ denotes the responses of nodes in the ith layer to the mth input vector.
9. The derivative of $F_{i:j}$ with respect to its scalar argument is denoted by either $f_{i:j}$ or $\sigma'_{i:j}$.
10. The nodes in the last layer can be either linear (in approximation problems) or nonlinear (in classification problems).
11. The biases for nodes in the ith layer are given by the s_i dimensional vector $\underline{b}_i = \{b_{i:j}\}$.
12. The weight $w_{i:j,k}$ assigned to the link connecting the kth node output in layer $i-1$ to the jth node input in layer i. Weights for each layer are elements of a $s_{i-1} \times s_i$ matrix W_i: $W_i = \{w_{i:k,j}\}$.
13. Vector $\underline{w}$ denotes all the parameters of the network, i.e. all the weights and all the biases.

Hence, in this notation the neural network operating equations are:

$$a_{0:j}^m = (\underline{x}_m)_j = x_j^m, \tag{3.10}$$

$$c^m_{i:j} = \sum_{k=1}^{s_{i-1}} w_{i:j,k} a^m_{i-1:k} + b_{i:j},\; i > 0, \tag{3.11}$$

$$a^m_{i:j} = F_{i:j}(c^m_{i:j}),\; i > 0. \tag{3.12}$$

The activation of a neuron in Lth layer is the output of the network:

$$a^m_{L:1} = y_m. \tag{3.13}$$

The error of the response to mth input is defined as follows:

$$\mathcal{E}_m(\underline{w}) = \frac{1}{2}(y_m - t_m)^2. \tag{3.14}$$

The entire error of approximation of a training set $\mathcal{T}$ is given by:

$$\mathcal{E}_{\underline{\mathcal{T}}}(\underline{w}) = \sum_{m=1}^{n} \mathcal{E}_m(\underline{w}). \tag{3.15}$$

Now we are ready to calculate partial derivatives in conformity with the chain rule of differentiation. Note that $w_{i:j,k}$ and $b_{i:j}$ affect only Equation 3.15 through their presence in Equation 3.11. Hence, the expression for all the elements of the gradient vector is

$$\frac{\partial \mathcal{E}_m}{\partial w_{i:j,k}} = \frac{\partial \mathcal{E}_m}{\partial c^m_{i:j}} \cdot \frac{\partial c^m_{i:j}}{\partial w_{i:j,k}} = \frac{\partial \mathcal{E}_m}{\partial c^m_{i:j}} \cdot a^m_{i-1:k}, \tag{3.16}$$

$$\frac{\partial \mathcal{E}_m}{\partial b_{i:j}} = \frac{\partial \mathcal{E}_m}{\partial c^m_{i:j}}. \tag{3.17}$$

Let $\delta^m_{i:j}$ denote the rate of contribution to the error $\mathcal{E}_m$ by excitation $c^m_{i:j}$ to the jth node in the ith layer:

$$\delta^m_{i:j} = \frac{\partial \mathcal{E}_m}{\partial c^m_{i:j}}. \tag{3.18}$$

This is the error signal "delta"that is backpropagated from the output to node j in layer i in response to the input $\underline{x}_m$. Equations 3.16 and 3.17 can be rewritten as follows:

$$\frac{\partial \mathcal{E}_m}{\partial w_{i:j,k}} = \delta^m_{i:j} \cdot a^m_{i-1:k}, \tag{3.19}$$

$$\frac{\partial \mathcal{E}_m}{\partial b_{i:j}} = \delta^m_{i:j}. \tag{3.20}$$

Since $\mathcal{E}_m$ depends on $c^m_{i:j}$ through $a^m_{i:j}$,

$$\delta^m_{i:j} = \frac{\partial \mathcal{E}_m}{\partial a^m_{i:j}} \cdot \frac{\partial a^m_{i:j}}{\partial c^m_{i:j}} = f_{i:j}(c^m_{i:j}) \cdot \frac{\partial \mathcal{E}_m}{\partial a^m_{i:j}}. \tag{3.21}$$

If layer i is hidden, then $\mathcal{E}_m$ depends on $a^m_{i:j}$ only through its effects on the layer $i+1$ to which it is an input. We have

$$\frac{\partial \mathcal{E}_m}{\partial a^m_{i:j}} = \sum_{k=1}^{s_{i+1}} \frac{\partial \mathcal{E}_m}{\partial c^m_{i+1:k}} \cdot \frac{\partial c^m_{i+1:k}}{\partial a^m_{i:j}} = \sum_{k=1}^{s_{i+1}} \delta^m_{i+1:k} \cdot w_{i+1:k,j}. \tag{3.22}$$

Substitution of Equation 3.22 into Equation 3.22 yields the *backward recursion* for $i < L$:

$$\delta^m_{i:j} = f_{i:j}(c^m_{i:j}) \sum_{k=1}^{s_{i+1}} \delta^m_{i+1:k} \cdot w_{i+1:k,j}. \tag{3.23}$$

To sum up, we will express the evaluation of gradient in the form of Algorithm 1.

Algorithm 1 Backpropagation algorithm for gradient evaluation

```
Require: $x_m$ – the input to the neural network, $\underline{b}_i$ – the vector of biases and $W \in \mathbf{R}^p$
    – the weights of the network
  1. Determine the node outputs $a^m_{i:j}$ and inputs $c^m_{i:j}$ in conformity with Equations 3.11
     and 3.12 in a forward pass of the training data through the network;
  2. By Equation 3.23 determine the $\delta^m_{i:k}$ in a backward pass through the network;
  3. Combine results to determine the gradient through Equations 3.19 and 3.20.
```

It is important to estimate the efficiency of the backpropagation algorithm, because it is intensively used in the process of neural network training. Let $p = \sum_{i=1}^{L} s_i(1 + s_{i-1})$ denote the number of parameters describing the network and let $s = \sum_{i=0}^{L} s_i$ indicate the number of neurons. We additionally assume that calculation of $F_{i:j}$ and $f_{i:j}$ needs $\phi = 3$ floating point operations (flops), because it requires three steps: division, addition, exponentiation. We also presume that the output function is liner.

forward pass No effort is desired in layer 0. In layer i, for each of s_i nodes, we need to evaluate $a^m_{i:j}$ and $c^m_{i:j} = F_{i:j}(c^m_{i:j})$. A computation of single $c^m_{i:j}$ involves s_{i-1} sums and s_{i-1} products, for a total of $2s_i s_{i-1}$ flops. Together with calculation of activation functions the flop count of the forward pass is:

$$fl_f = \sum_{i=1}^{L} s_i(2s_{i-1} + \phi) - \phi = 2p + s(\phi - 2) - \phi \approx 2p.$$

backward pass The calculation of $\sigma^m_{i:j}$ requires $s_{i+1} - 1$ additions and $s_{i+1} + 1$ multiplications, for a total of $2s_{i+1}$ flops for each of s_i such terms in layer i. The evaluation of $\delta^m_{L:1}$ requires only one substraction. Hence, the total number of flops required is

$$fl_b = 1 + 2\sum_{i=1}^{L-1} s_i s_{i+1} \approx 2p.$$

gradient calculation The gradient components for the weights require a single multiplication for each of the $p - s$ weights.

To sum up, the computation of gradient for a single input requires $fl = fl_f + fl_b + p - s \approx 5p$ flops. This certainly should by multiplied by the number of patterns in the training set and by the number of training iterations. It becomes clear, that the process of training of neural network requires high-speed computing, but it also indicates, that large, monolithic structures of networks should be decomposed. Such approach can yield a significant improvement of the learning process and performance of large neural networks.

3.3.2 Descent algorithms

The basic iterative recursion common to all the training methods widely used today, is a process that determines new parameters $\underline{w}_{k+1}$ in terms of the current values $\underline{w}_k$.

In *stochastic* approach, for each iteration, a training pattern is selected and the a *search direction* $\underline{d}_k$ and a *step size* α_k are determined. Afterwards the parameters $\underline{w}_k$ of the network are updated. This process is formulated as Algorithm 2. In this solution, the order in which the training patterns are chosen influences the process of training, because this order determines the path on the descending slope of the error function. There is a belief that this enables the algorithm to find better local minima through a more random exploration of the parameter space.

Algorithm 2 Stochastic descent algorithm of neural network training

```
Require: $\mathcal{T} = \{(\underline{x}_m, \underline{t}_m), m = 1 \ldots n\}$ – the training set, $K$ – maximal number of
  iterations, $\epsilon$ – tolerable error of training
  Initialize the parameters $\underline{w}$ of the network; $k := 1$;
  while $k \leq K$ and error of the network $\mathcal{E}_{\mathcal{T}}(\underline{w}_k) \geq \epsilon$ do
    Choose randomly $m$ such that $1 \leq m \leq n$;
    For $x_m$ and $t_m$ calculate search direction $\underline{d}_k$ and step size $\alpha_k$;
    Update parameters of the network: $\underline{w}_{k+1} = \underline{w}_k + \alpha_k \underline{d}_k$;
    $k := k + 1$;
  end while
```

An alternative approach (Algorithm 3), called *batch processing descent algorithm*, in contrast to Algorithm 2, cumulates the changes of the network parameters and updates weights and biases after the entire training set has been processed.

The key issue in both Algorithm 2 and 3 is to calculate proper values of the search direction $\underline{d}_k$ and the step size α_k, so that the error function $\mathcal{E}_{\underline{\mathcal{T}}}(\underline{w})$ is efficiently minimized. According to Equation 3.9, the best solution would be

$$\alpha_k \underline{d}_k = -H_k^{-1} \underline{g}_k.$$

Algorithm 3 Batch descent algorithm of neural network training

```
Require: $\mathcal{T} = \{(\underline{x}_m, \underline{t}_m), m = 1 \ldots n\}$ – the training set, $K$ – maximal number of epochs, $\epsilon$ – tolerable error of training
  Initialize the parameters $\underline{w}$ of the network; $k := 1$;
  while $k \leq K$ and error of the network $\mathcal{E}_{\mathcal{T}}(\underline{w}_k) \geq \epsilon$ do
    $\Delta\underline{w} := 0$;
    for all $m = 1 \ldots n$ do
      For $x_m$ and $t_m$ calculate search direction $\underline{d}$ and step size $\alpha$;
      $\Delta\underline{w} := \Delta\underline{w} + \alpha\underline{d}$;
    end for
    Update parameters of the network: $\underline{w}_{k+1} = \underline{w}_k + \Delta\underline{w}$;
    $k := k + 1$;
  end while
```

However, the calculation of the Hessian H and its inverse matrix H^{-1} is too computationally demanding and thus it is replaced with approximations.

The family of most often used descent algorithms are *steepest descent algorithms*. They use the negation of gradient to indicate the direction of parameters' changes:

$$\alpha_k > 0, \quad \underline{d}_k = -\underline{g}_k.$$

The parameter α_k should not be too large (since it may cause oscillations). However, when it is to small the convergence of learning is slow. It is often chosen adaptively. When the step of the iteration reduces the error, the value of α_k is slightly increased by a constant amount. Otherwise it is multiplied by the number slightly less than 1.

A direct application of the negation of the gradient for the descent direction may cause the stops of convergence in local minima. An algorithm with a momentum term may by a remedy for such problems, as it is a high-frequency filter. Now the change in parameter vector $\underline{w}$ depends not only on the current gradient but also on the most recent change in parameter vector:

$$\alpha_k > 0,\ 0 < \beta_k < 1, \quad \underline{w}_{k+1} = \underline{w}_k - \alpha_k \underline{g}_k + \beta_k \cdot (\underline{w}_k - \underline{w}_{k-1}).$$

There is a variety of other modifications of *gradient descent algorithm*. The one that proved very useful is a *conjugate gradient algorithm*. Its efficiency is caused by the way it chooses the search direction. In contrast to conventional gradient algorithm, it chooses such vector that is orthogonal to the directions previously explored and calculates the step size which minimizes the error function in the chosen direction. Certainly, after the error function has been minimized in all the dimensions of the parameters' space, the algorithm is restarted. The effectiveness of this approach is caused mainly by the fact that the successive directions of minimization do not spoil the results of the previous steps of the algorithm. A deeper study of this idea (as well as other solutions) and mathematical background can be found in [12, 38, 26].

It is necessary to emphasize that the parameters of a neural network should be appropriately initialized. If all of them equal zero no learning is possible, since the error is not propagated. Too large initial values cause the saturation on nonlinearities which results in flat areas of error function. Usually parameters are initialized at random with moderate or even small values (for example $[-0.2, 0.2]$). It is also possible to initialize them with application of a genetic algorithm or simulated annealing. This may cause faster convergence, since the initial solution is provided.

3.3.3 Information-theoretic error measures

The neural network is essentially an information-processing system in which randomness (noise) coexists with desired information. Hence the error function (cost function) can be defined in terms of relative entropy associated with the network parameters. The considerations of information and Shannon's entropy in Chapter 2 allow us to define the cross-entropy entropy error measure based on for example mutual or conditional information. Such attempts has been suggested and/or successfully made by [7, 51, 89]. Neelakanta in [51] performs a deep experimental study of the suitability of various information measures for neural network training. Some of his results will be mentioned below.

In the multilayer architecture of neural network, the minimization of the error function in the process of training refers to minimizing an energy functional (*Hamiltonian*). Such minimization of energy function can also be interpreted alternatively as the minimization of the state of disorganization caused by the noise present in the network. In thermodynamics energy and entropy are closely related quantities. Thus suggesting entropy-related error measures seem to be justifiable.

Neelakanta has implemented a multilayer perceptron with various distance measures used in the backpropagation algorithm. As the entropy related measures are non-negative, the sign of the direction of weight updates has been determined by target-output difference. He has observed faster convergence rate for some measures than the rate for the classical square-error function (SE). The best results has been achieved when *Rènyi* error measure

$$D_{RY}(p:q) = \mathcal{E}_{RY} = \frac{1}{\alpha - 1} \log \sum \left(p_i^{\alpha} q_i^{1-\alpha}\right)$$

and *Kapur 2* error measure

$$\begin{aligned} D_{KP(2)}(p:q) = \mathcal{E}_{KP(2)} &= \frac{\beta}{\alpha+\beta} \cdot \frac{1}{\alpha-1} \sum \log \left(p_i^{\alpha} q_i^{1-\alpha}\right) + \\ &+ \frac{\alpha}{\alpha+\beta} \cdot \frac{1}{\beta-1} \sum \log \left(p_i^{\beta} q_i^{1-\beta}\right) \end{aligned}$$

have been used. Moreover, for these measures, considerably better accuracy of approximation than for SE has been achieved.

3.4 Need for modularity

"The challenge of the next generation of neural networks is not learning by individual weight updating, but the composing of network modules and how to resolve the competition (and the cooperation) of the different modules." [65]

Although backpropagation neural networks prove very useful is contemporary science and engineering, there are many problems that we should be aware of. Many researchers call them "black box" matters. Actually, it is very difficult, or even impossible, to identify how and where inside the network the mathematical function describing the learning task is computed. (This function is certainly an approximation of the transfer function of the modelled system.) It reduces considerably the credibility of the neural network approach. Indeed, how to rely on the system if the internal representation of information is unknown. How to prove its correctness when it is demanded (for example in the case of a critical, dangerous control system)?

Another issue is a lack of clear indications what the structure of a neural network should be like. In the process of designing an application specific neural network, it is necessary to determine the learning algorithm, its parameters, the number of layers, the number of neurons in each layer, the activation function... Unfortunately, these decisions cannot be achieved in a systematical way, since there is no evidence what happens inside a neural network. As a consequence, these choices are always empirically achieved.

The next problem is pertinent to the most widely used neural network training algorithm — backpropagation algorithm. It is commonly known that its use leads quite often to slow learning convergence and/or learning failures. Since the backpropagation bases on finding local minima of error function by sliding down the steepest descent slop (Section 3.3), flat areas of the minimized function worsen the convergence of this algorithm. It obviously results in longer time of learning [30].

There is also a matter of interference inside the network. Suppose the network is to identify two disjoint concepts simultaneously (for example the trained problem is both spatial and temporal). It is very difficult for the network to extract the existence of this two different tasks and then to adapt itself to the whole task. This will cause the slow of convergence and the increase of error rate. [66]

And finally, suppose we have trained a neural network to perform a set of tasks and we want to extend its abilities and train a new task. In such case the backpropagation algorithm will forget the previously acquired knowledge.

Modularity is a remedy for most of the above mentioned problems. Note that this approach is widely used in computer science to simplify complicated problems and it is often expressed the "divide and conquer" paradigm. Indeed, if a neural system is built of several blocks, the neural subnetworks associated with them are definitely smaller and less complex that the monolithic neural network. Thus better convergence can be achieved. Secondly, the information flow in modular structure is clearer, because separate blocks are responsible for definite parts of computation. And finally, the computational (functional) abilities of a modular network can be easily extended by adding new units which are responsible for new tasks.

In Chapter 6 we will introduce the method of decomposition and the algorithms performing it. Basing on a priori knowledge of the system to be modelled on, we identify the independent parts of the model which are then trained separately and joined together as a modular system.

Chapter 4

Probabilistic analysis of learning

In Chapter 3 we have shown (Theorem 7) that single hidden layer neural network is capable of learning an arbitrary continuous multidimensional function. However, there remains the question what the structure of the network should be like. Precisely speaking, how many neurons in hidden layer (layers) should be present. In this chapter we will address this problem in the context of Probably Approximately Correct (PAC) model. We will introduce the Vapnik-Chervonenkis dimension (VC-dim) and other measures of neural network complexity. Also the relationship between neural network architecture and its processing abilities will be discussed. Finally we will refer the expected complexity of the network to the entropy of its input.

4.1 PAC model

In this section we will introduce *Probably Approximately Correct* (PAC) model of learning deviced by Valiant [83], Vapnik and Chervonenkis [84, 85]. The basic PAC model is applicable to neural networks with single, binary output. In other words, it is suitable for analysis of classification problems. In PAC it is assumed that neural network receives a stream of *examples* $\{x_i\} = \underline{x}$ with corresponding values of the *target* (*concept*) *function* $\{t(x_i)\}$, where $x_i \in \mathcal{X}$, $\mathcal{X}$ is a set of all possible inputs to the network and $t(.) : \mathcal{X} \to \{0, 1\}$ is a function to be learned which belongs to a set $\mathcal{C}$ of all possible target functions: $t(.) \in \mathcal{C}$. A fundamental assumption of PAC model is that the examples are presented to the network randomly, according to some distribution. Certainly, some examples may be more likely to be chosen than others.

To put it more formally, the training sample for $t(.)$ of length m is given in the following way:

$$\underline{s} = \Big((x_1, t(x_1)), (x_2, t(x_2)), \ldots, (x_m, t(x_m))\Big).$$

An example $x_i \in \mathcal{X}$ is said to be positive (negative) example of $t(.)$ if $t(x_i) = 1$ ($t(x_i) = 0$). Sometimes for short we will refer to a sequence $\underline{x} = (x_1, x_2, \ldots, x_m)$ as a *sample*. The set of all training samples of length m for $t(.)$ is denoted by $\mathcal{S}(m, t)$. As in has been described in Chapter 3, the learning algorithm accepts the training sample $\underline{s}$ and changes the parameters of the network, so that a better approximation of target function $t(.)$ is achieved.

Unfortunately, a finite neural network (with finite number of neurons) is not capable of processing an arbitrary function. Thus we have to define the *concept space* $\mathcal{C}$ (from which the target function $t(.)$ comes) and the *hypothesis space* $\mathcal{H}$ – the set of all functions computable by the network. The *learning algorithm* $L(.)$ is responsible for choosing a proper function $L(\underline{s}) \in \mathcal{H}$. Precisely speaking the $(\mathcal{C},\mathcal{H})$-*learning algorithm* is a function $L(.) : \bigcup_{m\geq 1, t\in\mathcal{C}} \mathcal{S}(m,t) \to \mathcal{H}$ from the set of all possible training samples for functions in $\mathcal{C}$, to the set $\mathcal{H}$ of all possible output hypotheses (functions calculated by a neural network).

To measure the quality of the solution achieved by the learning algorithm $h = L(\underline{s})$ we have to define the error measure. Since there is assumed to be some probability distribution μ on the set of all examples, we may define the *error* as follows:

$$er_\mu(h,t) = \mu\Big(\{x \in \mathcal{X} : h(x) \neq t(x)\}\Big),$$

where $\mathcal{X}$ is a set of all possible examples and $\mu(.)$ denotes a probability function $\mu(.) : 2^{\mathcal{X}} \to [0,1]$. When t is clear from the context the context we may use a simpler notation $er_\mu(h)$ instead of $er_\mu(h,t)$. We are ready to introduce Valiant's definition of PAC learning algorithm [83]:

Definition 15 (PAC learning algorithm) *Let L be a $(\mathcal{C},\mathcal{H})$-*learning algorithm*. Then L is probably approximately correct (PAC) if for any given ϵ and δ, $0 < \epsilon, \delta < 1$, there is a sample length $m_*(\delta,\epsilon)$ such that for all $t \in \mathcal{C}$ and for all probability distributions μ on the set of examples, we have*

$$m \geq m_*(\delta,\epsilon) \Rightarrow \mu^m\Big\{\underline{s} \in \mathcal{S}(m,t) : er_\mu(L(\underline{s}),t) > \epsilon\}\Big) < \delta,$$

where μ^m denotes the product probability distribution.

In other words, the algorithm is PAC if for each *accuracy parameter* ϵ and each *confidence parameter* there is $m_*(\delta,\epsilon)$ such that if sample is not shorter than $m_*(\delta,\epsilon)$ the probability that error of approximation exceeds ϵ is lower than δ. It is important to emphasize that the definition requires the minimal sufficient length of training sample to be independent of μ and t, depending only on δ and ϵ.

For better understanding of PAC notion we will consider the following example of PAC algorithm.

Definition 16 *The learning algorithm is* consistent *if given any training sample $\underline{s} = \Big((x_1,t(x_1)),(x_2,t(x_2)),\ldots,(x_m,t(x_m))\Big)$, the functions $L(\underline{s})$ and t agree on this sample: $\forall i = 1\ldots m \;\; L(\underline{s})(x_i) = t(x_i)$.*

Theorem 8 *If $\mathcal{H}$ is finite, any consistent $(\mathcal{H},\mathcal{H})$-learning algorithm is PAC.*

Proof The result of a consistent $(\mathcal{H},\mathcal{H})$-learning algorithm is a function $h \in \mathcal{H}$ which agrees on the training sample with the trained function $t(.)$. Let us consider all such functions $h \in \mathcal{H}$ whose error ϵ_k is greater than arbitrarily chosen ϵ. The probability that such function is consistent with $t(.)$ on a random sequence of length m is at most $(1-\epsilon)^m$. Since, $ln(z) \leq z - 1$, we have that $(1-\epsilon)^m \leq e^{-\epsilon m}$. There are at most $|\mathcal{H}|$ such functions ($\mathcal{H}$ is finite), so the probability that one of them is chosen equals at most $|\mathcal{H}| \cdot e^{-\epsilon m}$. As it should happen less often than δ we achieve $|\mathcal{H}| \cdot e^{-\epsilon m} < \delta$. From this inequality $m_*(\delta, \epsilon)$ can be easily derived:

$$m \geq m_*(\delta,\epsilon) = \frac{1}{\epsilon} \ln\left(\frac{|\mathcal{H}|}{\delta}\right),$$

a bound that is independent of both the distribution and the target function.

□

Is PAC learning possible when $\mathcal{H}$ is infinite?

4.2 Vapnik-Chervonenkis dimension

In this section we will present a theory which shows that PAC learning is also possible in cases whre hypothesis space $\mathcal{H}$ and concept space $\mathcal{C}$ are infinite.

Suppose $\mathcal{F}$ is a set of $\{0,1\}$-valued functions defined on set $\mathcal{X}$, and again let $\underline{x} = (x_1, x_2, \ldots, x_m)$ be a sample o length m of examples from $\mathcal{X}$.

Definition 17 *The* number of classifications $\Pi_{\mathcal{F}}(\underline{x})$ *of* $\underline{x}$ *by* $\mathcal{F}$ *is the number of distinct vectors of the form*

$$\underline{x}^*(f) = (f(x_1), f(x_2), \ldots, f(x_m)), \quad f \in \mathcal{F}.$$

Note, that although $\mathcal{F}$ may by infinite, $\Pi_{\mathcal{F}}(\underline{x})$ is always finite and upper bounded by the number of all possible classifications of sample $\underline{x}$ of length m, $\Pi_{\mathcal{F}}(\underline{x}) \leq 2^m$.

An important quantity for learning theory is the *growth function* defined as the maximum possible number of distinct classifications by $\mathcal{F}$ of a sample of a fixed length m.

Definition 18 (The growth function) *The* growth function $\Pi_{\mathcal{F}}(m)$ *is the maximum possible number of classifications by $\mathcal{F}$ of a sample of a given length m:*

$$\Pi_{\mathcal{F}}(m) = \max\{\Pi_{\mathcal{F}}(\underline{x}) : \underline{x} \in \mathcal{X}^m\}.$$

Again, $\Pi_{\mathcal{F}}(m) \leq 2^m$. If this upper bound is attained for a certain sample $\underline{x} \in \mathcal{X}^m$, we say that the sample $\underline{x}$ is shattered *by $\mathcal{F}$.*

To get a better understanding of the above definition, let us consider the following examples. (Examples 2 and 3 are from [64].)

Example 2 *Consider the concept class of intervals* $[0, a]$ *on the real line,* $0 < a < 1$. *Clearly samples of size 1 can be shattered by this class, since if we pick some point* x *as our sample, the point* a *can be placed to the left or the right, thus excluding or including* x. *However, no sample of size* 2 *can be shattered, since it is impossible to choose* a *such that* x_2 *is included and* x_1 *is excluded if* $x_2 > x_1$.

Example 3 *Consider the class of subintervals* $[a, b]$, $0 < a, b < 1$. *Here a sample of length* 2 *is shattered, but no sample of size* 3 *is shattered, since no concept can satisfy a sample which middle point is negative and outer points are positive.*

Example 4 *Let* $\mathcal{X} = \mathbf{R}^2$ *is a set of all points in the plain and* $\mathcal{F}$ *is a family of such functions that* $f(x) = \begin{cases} 1, & a^T x + b \geq 0 \\ -1, & a^T x + b < 0 \end{cases}$, *where* $a \in \mathbf{R}^2$ *and* $b \in \mathbf{R}$. *A set of* $m = 3$ *arbitrarily chosen points such that they do not lay on the same line is shattered by* $\mathcal{F}$ *(Figure 4.1). In the case of* $m = 4$ *points shattering is impossible, because there is no such function* $f \in \mathcal{F}$ *that* $f(x_1) = f(x_3) \wedge f(x_2) = f(x_4)$ *(Figure 4.2). We can count elementarily that* $\Pi_{\mathcal{F}}(4) = 14$. *(The linear-separability problem will be studied more deeply in the later part of this chapter.)*

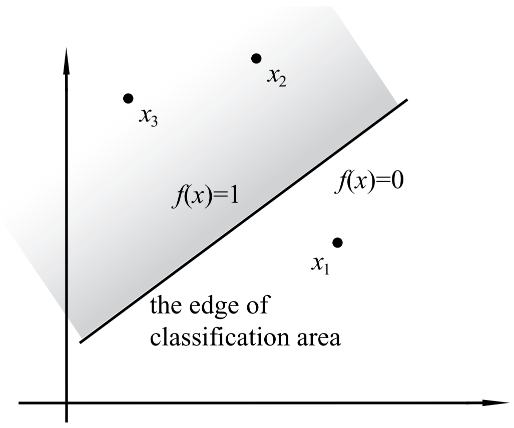

Figure 4.1: The shattering of three points with half-spaces in $\mathbf{R}^2$ is possible.

Based on the intuitive notion that a set $\mathcal{F}$ of functions is effective in classification if it can achieve all the possible classifications of a large set of examples, we can introduce the Vapnik-Chervonenkis dimension (VC-dim). The VC-dimension of $\mathcal{F}$ is the maximum length m of a sample $\underline{x} \in \mathcal{X}^m$ shattered by $\mathcal{F}$:

$$\mathrm{VCdim}(\mathcal{F}) = \max\{m : \Pi_{\mathcal{F}}(m) = 2^m\}$$

If there is no such maximum, we say that the VC-dimension of $\mathcal{F}$ is infinite. We state this definition more formally.

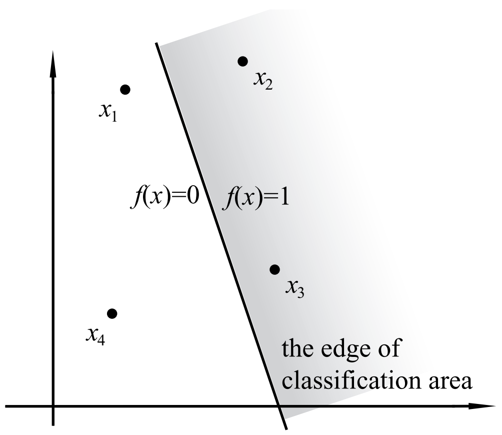

Figure 4.2: The shattering of four points with half-spaces in $\mathbf{R}^2$ is impossible.

Definition 19 (The VC-dimension)
Let $\mathcal{F}$ be a set of functions from a set $\mathcal{X}$ to $\{0,1\}$. The VC-dimension of $\mathcal{F}$ is (infinite, or) the maximal size of a subset $\mathcal{E} \subseteq \mathcal{X}$ such that for each $\mathcal{S} \subseteq \mathcal{E}$, there is $f_{\mathcal{S}} \in \mathcal{F}$ with $f_{\mathcal{S}}(x) = 1$ if $x \in \mathcal{S}$ and $f_{\mathcal{S}}(x) = 0$ if $x \in \mathcal{E} \setminus \mathrm{S}$.

Now we can identify the VC-dim of the concept spaces in Examples 2, 3 and 4, which equal 1, 2 and 3, respectively.

Suppose we have a binary output neural network $\mathcal{N}$ with a set $\mathcal{X}$ of all possible inputs. The hypothesis space $\mathcal{H}$ of the network is the set of all functions computable by the network on example space $\mathcal{X}$. According to Definition 19, the VC-dim($\mathcal{N}$) of such neural networks is the maximal size of the example set which is shattered by $\mathcal{N}$. Speaking more formally:

Definition 20 (The VC-dimension of a neural network)
Let $\mathcal{N}$ be a binary-output neural network capable of taking on a number of states (determined by its parameters), and let Ω denote the set of all possible states. Suppose that $\mathcal{X}$ is a set of all possible inputs. Let $\mathcal{N}_\omega$ be the function from $\mathcal{X}$ to $\{0,1\}$ computed by $\mathcal{N}$ when its state is ω, and let

$$\mathcal{H}_{\mathcal{N}} = \{\mathcal{N}_\omega : \omega \in \Omega\}$$

be the set of functions computable by $\mathcal{N}$. Then the VC-dimension of $\mathcal{N}$ on example set $\mathcal{X}$, denoted $\mathrm{VCdim}(\mathcal{N}, \mathcal{X})$, is defined to be $\mathrm{VCdim}(\mathcal{H}_{\mathcal{N}})$.

Certainly, if the set of all possible states Ω of the network is finite (as is in case of any digitally simulated network), the set of all computable by network function $\mathcal{H}_{\mathcal{N}}$ is also finite. Since the maximal number of dichotomies cannot exceed $|\mathcal{H}_{\mathcal{N}}|$, $\mathrm{VCdim}(\mathcal{N}, \mathcal{X})$ is finite and does not exceed $\log(|\mathcal{H}_{\mathcal{N}}|)$.

It is interesting to notice that computational abilities of a set of functions $\mathcal{F}$ are unequivocally determined by CVdim($\mathcal{F}$). Vapnik and Chervonenkis discovered [85] that maximal number of dichotomies on a set of n examples equals 2^n as n increases from zero up to a certain point (n =CVdim($\mathcal{F}$)). After that point growth slows and can by bounded by a polynomial function of CVdim($\mathcal{F}$) [11, 64]. We will state it more precisely in the following theorem.

Theorem 9 (Sauer's Lemma) *Let $d \geq 0$ and $m \geq 1$ be given integers and let $\mathcal{F}$ be a set of $\{0,1\}$-valued functions with* $\text{VCdim}(\mathcal{F}) = d \geq 1$. *Then*

$$\Pi_{\mathcal{F}}(m) \leq \sum_{i=0}^{d} \left(\begin{array}{c} m \\ i \end{array} \right) < \left(\frac{em}{d}\right)^d,$$

where the second inequality holds for $m \geq d$, $e = 2.718\ldots$

Proof
(1) Let $\Phi_d(m) = \left\{ \begin{array}{l} 1 \text{ if } d=0 \text{ or } m=0 \\ \Phi_d(m-1) + \Phi_{d-1}(m-1) \end{array} \right.$.
It turns out that $\Phi_d(m) = \sum_{i=0}^{d} \left(\begin{array}{c} m \\ i \end{array} \right)$ and we will prove it by induction.
Base cases:
if $d = 0$, $\left(\begin{array}{c} m \\ 0 \end{array} \right) = 1$
if $m = 0$, $\sum_{i=0}^{d} \left(\begin{array}{c} 0 \\ d \end{array} \right) = \left(\begin{array}{c} 0 \\ 0 \end{array} \right) = 1.$
Inductive step:

$$\begin{aligned}
\Phi_d(m) &= \Phi_d(m-1) + \Phi_{d-1}(m-1) \\
&= \sum_{i=0}^{d} \left(\begin{array}{c} m-1 \\ i \end{array} \right) + \sum_{i=0}^{d-1} \left(\begin{array}{c} m-1 \\ i \end{array} \right) \\
&= \sum_{i=0}^{d} \left[\left(\begin{array}{c} m-1 \\ i \end{array} \right) + \left(\begin{array}{c} m-1 \\ i-1 \end{array} \right) \right] \\
&= \sum_{i=0}^{d} \left(\begin{array}{c} m \\ i \end{array} \right)
\end{aligned}$$

(2) If $m > d$ then $0 \leq \frac{d}{m} < 1$. We can write:

$$\begin{aligned}
\left(\frac{d}{m}\right)^d \sum_{i=0}^{d} \left(\begin{array}{c} m \\ i \end{array} \right) &\leq \sum_{i=0}^{d} \left(\frac{d}{m}\right)^i \left(\begin{array}{c} m \\ i \end{array} \right) \leq \\
&\leq \sum_{i=0}^{m} \left(\frac{d}{m}\right)^i \left(\begin{array}{c} m \\ i \end{array} \right) = \left(1 + \frac{d}{m}\right)^m \leq e^d.
\end{aligned}$$

Dividing both sides by $\left(\frac{d}{m}\right)^d$, for $m > d$, we get:

$$\Phi_d(m) \leq e^d \left(\frac{m}{d}\right)^d = \left(\frac{me}{d}\right)^d .$$

(3) Now we will show that $\Pi_{\mathcal{F}}(m) \leq \Phi_d(m)$, where $d = \mathrm{VCdim}(\mathcal{F})$, and is therefore polynomially bounded. This step of prove proceeds by double induction on m and d.

Let us consider the base cases.
When $m = 0$, there can only be one subset, hence $\Pi_{\mathcal{F}}(0) \leq 1 = \Phi_d(0)$.
When $d = \mathrm{VCdim}(\mathcal{F}) = 0$, no set of points can be shattered, hence all points can be labelled only one way. From this we conclude that $\Pi_{\mathcal{F}}(m) = 1 \leq \Phi_0(m)$.

Now suppose that the induction hypothesis is true up to $m-1$, i.e. $\Pi_{\mathcal{F}}(m-1) \leq \Phi_d(m-1)$. It means there exists a set $\mathcal{S}_{m-1}$ of $m-1$ elements having $\Pi_{\mathcal{F}}(m-1)$ such subsets, that each subset can be labelled by a certain function from the set $\mathcal{F}$ in the same way.
(a) Now we add one point x to the set $\mathcal{S}_{m-1}$: $\mathcal{S}_m = \mathcal{S}_{m-1} \cup \{x\}$. It increases the maximal number of subsets which can be labelled in the same way. This additional subsets are certainly the subsets which contain x. The question is what is the number (at most) of such subsets? (b) Since x is present in each such subset, the number of free elements for combinations is $m-1$. (c) $d_{m-1} = \mathrm{VCdim}(\mathcal{F})$ means that $\mathcal{F}$ can shatter at most a set of d_{m-1}. Since x is always labelled, $d_m = \mathrm{VCdim}(\mathcal{F}) - 1$. According to (b) and (c) we have that this additional number is at most $\Phi_{d-1}(m-1)$. All together we get:

$$\Pi_{\mathcal{F}}(m) \leq \Pi_{\mathcal{F}}(m-1) + \Phi_{d-1}(m-1) \leq \Phi_d(m-1) + \Phi_{d-1}(m-1) = \Phi_d(m)$$

$\square$

Figure 4.3 depicts an example of the upper bound for the growth function of a set of functions $\mathcal{F}$ such that $\mathrm{VCdim}(\mathcal{F}) = 10$. The function $\Pi_{\mathcal{F}}(m)$ growths exponentially as 2^m until m reaches 10. After that, growth considerably slows. It clearly demonstrates that when the size of the input set is too large, the system, whose VC-dimension is finite (for example digital neural network), looses its processing capabilities. In other words, the amount of information that such system can arbitrarily process is also finite and is in some way bounded by VC-dimension. As it is depicted in Figure 4.4, if $m > \mathrm{VCdim}(\mathcal{F})$ the number of functions computable by the system is considerably less than the number of all distinct functions. It clearly shows that VC-dimension is a good measure of computational capabilities of the set of functions.

We have motivated our discussion of VC-dimension by describing it as a measure of the expressive power of a set of functions. Right now we will find the VC-dimension of sets of functions which are important in neural network computing.

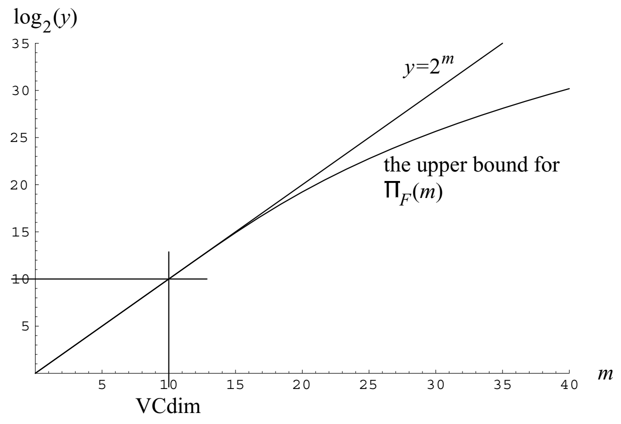

Figure 4.3: The upper bound for the growth function $\Pi_{\mathcal{F}}(m)$, where $\mathrm{VCdim}(\mathcal{F}) = 10$.

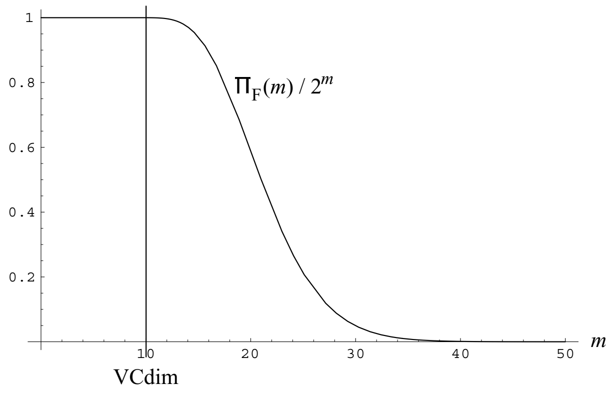

Figure 4.4: The number of distinct binary-output functions computable by $\mathcal{F}$ in relation to all computable functions, where m is a size of a sample set and $\mathrm{VCdim}(\mathcal{F}) = 10$.

4.3 VC-dimension of a linear perceptron

In this section we will determine the VC-dimension of a threshold perceptron and generally of a linear space of functions. First of all we will begin with a helpful theorem.

Theorem 10 *A set $\mathcal{E} = \{x_1, x_2, \dots, x_k\}$ of points of ^{n} can be shattered by the set $\mathcal{F}$ of such functions that $f(x) = \begin{cases} 1 & if \quad w^Tx > 0 \\ -1 & if \quad w^Tx \le 0 \end{cases}$, $w \in \mathbf{R}$, if and only if the points of $\mathcal{E}$ are linearly independent.*

Proof
(1) Since there are at most $k = n$ linearly independent points $x_1, x_2, \dots, x_n$ of $\mathbf{R}^n$, we will show that the set of n linearly independent n-dimensional points can be shattered. Let A be the matrix whose n rows consist of these linearly independent points (as vectors) and let $v \in \{0,1\}^n$. The matrix A is nonsingular and can by inverted. Thus for all $v \in \{0,1\}^n$ there exists $w = A^{-1}v \in \mathbf{R}^n$ such that $Aw = v$. Thus all classifications of the set of points are possible and the set is shattered.
(2) It is necessary to demonstrate that a set of linearly dependent points cannot be shattered. In such set at least one point is a linear combination of the others. Without loss of generality let it be x_1: $x_1 = \sum_{i=2}^k \lambda_i x_i$, where at least one parameter $\lambda_i \neq 0$. Consider the following classification of points x_j, $j \neq 1$. Let x_j be classified positively ($w^Tx_j > 0$) if $\lambda_j > 0$ and negatively ($w^Tx_j \le 0$) otherwise. This classification determines the sign of $w^Tx_1 = \sum i = 2^k \lambda_i w^T x_i > 0$, thus all distinct classifications are not possible and the set of linearly dependent points cannot be shattered.

$\square$

And now we can determine the VC-dimension of the perceptron, or a separate neuron (Figure 4.5) — a fundamental component of a threshold neural networks. For a positive

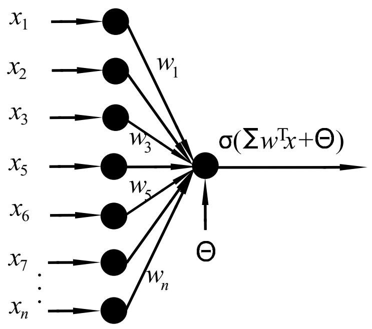

Figure 4.5: The perceptron (neuron).

integer n, let P_n be a the threshold perceptron, which has n inputs and a single computation unit. The arcs carrying the inputs have real-valued weights $w_1, w_2, \dots, w_n$

and there is a threshold value θ at the active unit. $\sigma(.)$ is a Heaviside function: $\sigma(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ -1 & \text{if} \quad x \leq 0 \end{cases}$. The output equals $\sigma(w^T x + \theta)$. We have the following theorem.

Theorem 11 *Let P_n be a threshold perceptron on n inputs. Then*

$$\mathrm{VCdim}(P_n, \mathbf{R}^n) = n + 1 \textit{ and } \mathrm{VCdim}(P_n, \{0,1\}^n) = n + 1.$$

Proof Let us transform the problem to homogeneous coordinates:

$$\begin{aligned} \sigma(w^T x + \theta) &= \sigma\Big(\theta + (w_1, w_2, \ldots, w_n)^T (x_1, x_2, \ldots, x_n)\Big) \\ &= \sigma\Big((\theta, w_1, w_2, \ldots, w_n)^T (1, x_1, x_2, \ldots, x_n)\Big) \end{aligned}$$

The space in both is cases n dimensional so we can find n linearly independent points (vectors) $v_1, v_2, \ldots, v_n$. Let v_{n+1} be a linear combination of $v_1, v_2, \ldots, v_n$: $v_{n+1} = \sum_{i=1}^n \lambda_i v_i$, such that $\sum_{i=1}^n \lambda_i \neq 1$. Let $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_n, \tilde{v}_{n+1}$ denote the set of augmented vectors such that $\tilde{v}_i = (1, v_i) = (1, v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(n)})$, where v_i^j indicates the jth component od the ith vector. The vectors $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_n$ are linearly independent because $v_1, v_2, \ldots, v_n$ are independent. By $\sum_{i=1}^n \lambda_i \neq 1$ there is no such linear combination that $\tilde{v}_{n+1} = \sum_{i=1}^n \lambda_i \tilde{v}_i$. Hence, the set $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_n, \tilde{v}_{n+1}$ is linearly independent and by Theorem 10 can be shattered. Certainly, $n + 1$ is the largest size of linearly independent vectors in $n + 1$ dimensional space. Thus in both cases VCdim $= n + 1$.

$\square$

Now we know the computational abilities of a single threshold neural unit and we have formally achieved the answer to the problem addressed in Example 4. Unfortunately, it is substantially harder to calculate the VC-dimension for more complicated neural networks. In the later part the bounds for VCdim of various neural architectures will be given. Proofs will be mostly omitted because they often involve deep mathematical concepts.

4.4 VC-dimension of neural networks

It is extremely hard to give an exact VC-dimension of a neural network. It is caused by the fact that such structures are highly nonlinear and thus simple algebraic notions cannot be applied. Secondly, VC-dimension of neural network strictly depends on its architecture. Such exact result for particular neural network would not be general and thus it would be useless.

4.4.1 Threshold neural networks

Now we present the upper bounds on VC-dimension of linear threshold neural networks, which consist of linear neurons (like in Figure figure:perceptron) with Heaviside activation function $\sigma()$ as before.

The following result, a bound on VC-dimension of such networks has been given by Natarajan [50].

Theorem 12 *Let $\mathcal{N}$ be a linear threshold network with N neurons (including the input nodes), n input nodes and a total of W variable weights and thresholds. Then*

$$\mathrm{VCdim}(\mathcal{N}, \{0,1\}^n) \leq WN \log N.$$

Proof can be found in [50].

□

Unfortunately, the above theorem is limited only to binary input threshold neural networks. The following result, by Baum and Haussler [43], gives a bound on VC-dimension of real input neural networks and improves Natarajan's result for binary input networks.

Theorem 13 *Let $\mathcal{N}$ be a feedforward linear threshold network having a total of W variable weights and thresholds, and n inputs. Then*

$$\mathrm{VCdim}(\mathcal{N}, \{0,1\}^n) \leq \mathrm{VCdim}(\mathcal{N}, \mathbf{R}^n) \leq 6W \log W.$$

Proof can be found in [43].

□

In the later part of this subsection the lower bounds of VC-dimension are given. Before we present the results, we remind same basic asymptotic notions. Let $g(x)$ and $f(x)$ be functions on $x \in \mathbf{R}$, such that $g(x), f(x) > 0$. We say that

- $g(x)$ is of order of $f(x)$, which is denoted as $g(x) = \Theta(f(x))$, when $\lim_{x\to\infty} \frac{g(x)}{f(x)} = c$, $c \in \mathbf{R}$, $c < \infty$;
- $g(x)$ is at least of order of $f(x)$, which is denoted as $g(x) = \Omega(f(x))$, when $\exists c > 0\ \forall x > 0\ g(x) \geq cf(x)$;
- $g(x)$ is at most of order of $f(x)$, denoted as $g(x) = \mathrm{O}(f(x))$, if $f(x) = \Omega(g(x)$;
- $g(x)$ is of lower order than $f(x)$, denoted as $g(x) = \mathrm{o}(f(x))$, if $\lim_{x\to\infty} \frac{g(x)}{f(x)} = 0$.

Theorem 14 *Assume that $\mathcal{N}_n$ is any sequence of feedforward linear threshold networks of depth at least three (having two hidden layers), fully connected between successive layers, having n inputs, and such that $\mathcal{N}_n$ has $\Omega(n)$ threshold units in the first hidden layer and at least $4\log(n)$ units in the second hidden layer. Then*

$$\mathrm{VCdim}(\mathcal{N}_n, \{0,1\}^n) = \Theta(n^2 \log(n)).$$

Proof can be found in [44].

$\square$

Note that, since the network in Theorem 14 is fully connected and the result is tight, we have that $\mathrm{VCdim}(\mathcal{N}_n, \{0,1\}^n) = \Theta(W\log(W))$, where W is the total number of weights and thresholds. This result is consistent with Theorem 13. And what is in the case of real input networks? The results are slightly better, since the network achieved by Sakurai [69] has only one single layer.

Theorem 15 *Let $\mathcal{N}$ be a fully connected depth-two linear threshold network having n real inputs and h units in the hidden layer. Then*

$$\mathrm{VCdim}(\mathcal{N}_n, \mathbf{R}^n) \geq \frac{1}{2}nh\left(\log h + o(\log h) + O\left(\frac{\log^2 h}{n}\right)\right)$$

and

$$\mathrm{VCdim}(\mathcal{N}_n, \mathbf{R}^n) \leq nh(\log h + o(\log h)),$$

as $h, n \to \infty$.

Proof can be found in [69].

$\square$

To sum up, in the case of threshold neural networks VC dimension is $O(W \log W)$.

4.4.2 Sigmoid neural networks

In this section more complex types of neural networks will be discussed. The bounds obtained in the previous subsection were quite tight. However, in the case of more sophisticated activation functions, the results are less precise.

We will begin the discussion with the result of Sontag [78]. He has shown that there is a neural network $\mathcal{N}$ of infinite VC-dimension, having only two real inputs and and two hidden units with sigmoid activation function given by the following equation:

$$\sigma(x) = \frac{1}{\pi}\arctan(x) + \frac{\cos(x)}{\alpha(1+x^2)} + \frac{1}{2},\ \alpha > 2\pi.$$

The output neuron is a standard threshold perceptron of 4 and threshold, since direct connections between input and output are allowed. This is rather a simple network but one of infinite capabilities. In view of this, it is impossible to obtain a general VC-dimension bound for sigmoid neural networks and each case should be studied separately. Generally, the sigmoid networks have better computational capabilities that those with threshold activation function. It is caused by the fact that any single sigmoid function (Definition 14) can arbitrarily precisely approximate the step function (Heaviside function) as it has been shown in the proof of Lemma 1. Thus the lower bounds of VC-dimension for threshold networks are also valid for sigmoid networks.

The question is whether there are sigmoid networks of finite VC-dimension? The answer is given in the theorem of Macintyre and Sontag [45].

Theorem 16 *Let $\mathcal{N}$ be a feedforward network with binary output and the logistic function as activation function on the hidden units. Then $\mathcal{N}$ has finite VC-dimension.*

Proof can be found in [45].

□

Moreover, there is a polynomial upper bound on VC-dimension of neural networks with logistic activation functions [35].

Theorem 17 *Let $\mathcal{N}$ be a feedforward network with binary output and the logistic function as activation function on the hidden units. Suppose that the total number of adjustable weights and thresholds is W and that there are k computational units. Then* $\mathrm{VCdim}(\mathcal{N}) \leq \frac{Wk(Wk-1)}{2} + W(2k+1)\log(2d) + W(3k+1)\log((2k+1)W+1) + W(3k+1)\log(2d+1)$*, where d is a certain positive constant. This implies that*

$$\mathrm{VCdim}(\mathcal{N}) = O(W^2k^2) = O(W^4).$$

Proof can be found in [35].

□

We end this subsection with a very important result settling a long-standing open question whether the $O(W \log W)$ bound for threshold networks also holds for sigmoidal nets. Koiran and Sontag [37] present the following result.

Theorem 18 *Let $\mathcal{N}$ be a single binary output feedforward neural network with an activation function σ which has different limits at $\pm\infty$, and is such that there is at least one point where it has a derivative and the derivative is nonzero (this rules out the Heaviside activation). Let $\mathcal{N}$ has W variable parameters. We have that*

$$\mathrm{VCdim}(\mathcal{N}) = \Omega(W^2).$$

Proof can be found in [37]. The proof relies on first showing that networks consisting of two types of activation functions, Heaviside's and linear $\sigma(x) = s$, already have this power. However, Heaviside's function can be approximated with sigmoid function and large weights, whereas and linear activation can be approximated with sigmoid function and small weights (that is why the existence of derivative is needed).

□

In conclusion, there is still a large gap between $O(W^4)$ and $\Omega(W^2)$ — the bounds on VC-dimension of sigmoid neural networks.

4.5 VC-dimension and PAC learning

In this section we will be more specific about algorithms. Is neural network training algorithm L PAC? We may assume that the hypothesis space $\mathcal{H}$ (the set of all functions computable by the network) equals the concept space $\mathcal{C}$ (the set of all functions of intrest). We may also assume that this training algorithm L is consistent, which means that it can train an arbitrary sample $\underline{s}$ of an arbitrary length m of any function from $\mathcal{C} = \mathcal{H}$ without errors. If additionally $\mathrm{VCdim}(\mathcal{H}) = d < \infty$, by Theorem 19, L is PAC. Moreover, Theorem 19 gives the relationship between *sample complexity* m_L, which is the least integer than can be taken as $m_*(\delta, \epsilon)$) in the definition of PAC learning, and VC-dimension of $\mathcal{H}$. In other words, sample complexity indicates the sufficient number of elements of the training set sufficient to achieve probably approximately correct result of training. The following theorem is a result of Vapnik.

Theorem 19 *Suppose that the concept space $\mathcal{C}$ is contained in the hypothesis space $\mathcal{H}$. Suppose that $\mathcal{H}$ has finite VC-dimension. Let L be any consistent $(\mathcal{C}, \mathcal{H})$-learning algorithm. Then L is PAC, and its sample complexity m_L satisfies the inequality*

$$m_L(\delta, \epsilon) \leq \frac{8}{\epsilon}\left(\ln\left(\frac{4}{\epsilon}\right) + \mathrm{VCdim}(\mathcal{H}) \cdot \ln\left(\frac{48}{\epsilon}\right)\right).$$

Proof can be found in [11].

□

We clearly see, that if VCdim is finite then the sample complexity is upper bounded. This is an important result since it indicates that, independently of the function being trained, for arbitrarily good approximation there is a sufficient size of a training set. On the other hand, according to Blumer [11], if $\mathrm{VCdim}(\mathcal{H})$ is infinite then PAC learning is impossible.

4.6 Generalizations of VC-dimension

4.6.1 Sontag dimension

We have defined the VC dimension as the largest k such that some k-element set is shattered. In general, one cannot shatter all sets of that size. For example as it has been already shown a set of 3 points in the 2-dimensional plain can be shattered by half-spaces in $\mathbf{R}^2$ only on condition that they do not lay on the same line. However the VCdim of the set of half-spaces is 3. This leads to the question: when can one shatter "generic" sets of size equal to the VC dimension? Sontag in [78] suggests the following measure of the computation power of the set of functions.

Definition 21 *Let $\mathcal{X}$ be an input set and the set $\mathcal{S}_k \subset \mathcal{X}^k$ consist of the ordered sets of k inputs which can be shattered. Sontag $\underline{\mu}$ dimension is defined as follows:*

$$\underline{\mu} = \sup\{k \geq 1 : \mathcal{S}_k \textit{ is a dense subset of } \mathcal{X}^k\}.$$

As an example, for perceptrons we have $\underline{\mu}$ = VCdim = $m + 1$. Generally, any time that one has a vector space of analytically parametrized class of functions, if the VC dimension is k then generic sets of that size can be shattered, as simply a determinant must be nonzero, so for such linear classes one gets equality of VC dimension and $\underline{\mu}$. However, $\underline{\mu}$ may be strictly less that the VC dimension [77] and generally $\underline{\mu}(\mathcal{H}) \leq \text{VCdim}(\mathcal{H})$.

4.6.2 Graph dimension

In this section we will introduce a measure which is a direct consequence of VC dimension, but is suitable for multivalued multiple output neural networks.

Suppose, as before, that $\mathcal{H}$ and $\mathcal{C}$ are sets of functions from an example space $\mathcal{X}$ into a set $\mathcal{Y}$ (not necessarily {0,1}) with $\mathcal{C} \subseteq \mathcal{H}$, and suppose $t(.) \in \mathcal{C}$. The error of $h(.) \in \mathcal{H}$ with respect to $t(.)$ is

$$\text{er}_\mu(h, t) = \mu(\{x \in X : h(x) \neq t(x)\}),$$

where μ is a probability distribution on $\mathcal{X}$. (Obviously, this measure is not suitable for the situation when $\mathcal{Y}$ is a dense set.)

Now let $Gh : X \times Y \to \{0, 1\}$, called the *graph* of $h(.)$, be the function defined by

$$Gh(x, y) = 1 \Leftrightarrow h(x) = y.$$

The set of such functions on $\mathcal{H}$: $G\mathcal{H} = \{Gh : h \in \mathcal{H}\}$ is called the *graph space* of $\mathcal{H}$. Natarajan [49] has formulated the following definition:

Definition 22 *For a set $\mathcal{H}$ of functions from $\mathcal{X}$ to $\mathcal{Y}$, the* graph dimension *of $\mathcal{H}$, denoted* Gdim($\mathcal{H}$)*, is the VC dimension of the set $G\mathcal{H}$.*

If the graph dimension of a set of functions $\mathcal{H}$ is d it means that there is a set of d inputs $x_1, x_2, \ldots, x_d$ that can be classified in an exhaustive way:

$$\forall y_1, y_2, \ldots, y_d \in \mathcal{Y}\ \exists f(.) \in \mathcal{H} \quad (f(x_1), f(x_2), \ldots, f(x_d)) = (y_1, y_2, \ldots, y_d).$$

Using Definition 22 we can consider multiple output and multiple valued neural networks in terms of VC dimension and PAC learnability.

4.7 Neural network dimension and the entropy of input

In Chapter 3 we have shown that the time complexity of backpropagation algorithm is $O(np^2)$, where n denotes the number trained patterns and p stands for the number of adjustable parameters of the neural network. We can shorten the time of training by reducing p. Unfortunately, as it has been proved in this chapter, the lower the number of weights is the weaker the computational abilities of the network are. It directly flows from the fact that VC dimension is upper bounded by the function of the number of weights in the network. VC dimension as well as graph dimension denote the maximal cardinality d of a certain set $\mathcal{Z}_d$ of symbols that can be arbitrarily processed by the network. It has a straightforward impact on the amount of information that can be processed. Let these symbols form an alphabet $\mathcal{Z}_d$ with corresponding probability of each symbol. According to Theorem 1, the entropy H of such random variable Z reaches maximum when the distribution of symbols of the alphabet is uniform. Thus the entropy is upper bounded:

$$\begin{aligned} \mathrm{H}(Z) &\leq \log(VCdim(\mathcal{N}, \mathcal{X})), \\ \mathrm{H}(Z) &\leq \log(Gdim(\mathcal{N}, \mathcal{X})). \end{aligned}$$

Thus, if we want to reduce the time of learning by reducing the size of network, an evident solution is decomposition since each block of the decomposed network performs a simpler task and is supported only with partial information of lower entropy than the entropy of the whole input. The lower the entropy of the processed information is the smaller the size of the network and thus the shorter time of training.

Chapter 5

Neural network input space reduction

In the area of machine learning, artificial intelligence, logic synthesis and finally neural networks, we often deal with data having functional dependencies or carrying information that is not necessary for the computation. For example in order to analyze multi-dimensional input vectors of, say, hundreds of different stock prices, it would be easier to work with the most critical and representative features of data. In such cases not all attributes or inputs are necessary and some of them can be simply removed without loss of information. Sometimes the part of input information unnecessary for computation can be removed. As it has been shown in the previous chapters, the unnecessary information causes that bigger structures of neural networks are needed. It results in longer time of training.

First of all we will identify two main types of input redundancy:

Functional dependencies Such situation takes place when one input can be directly computed from the others and thus this input does not bring additional information.

Unnecessary information Sometimes information provided by some of inputs is not needed in the process of computation, because the output value does not depend on it. The situation is quite often in Machine Learning when the designer of the system is unaware which inputs actually influence the results.

In this chapter the approaches capable of reducing the input information will be discussed. I will begin with two methods widely applied in the area of neural networks and data analysis.

5.1 Principle Component Analysis

Principal Component Analysis, or PCA [33], also known as Hotelling transform and Karhunen-Loeve transform, is widely used in signal processing, pattern recognition and neural computing. It is also considered as a means of compression, namely algebraic compression.

The idea of PCA is to find such orthogonal directions of data cloud such that the variance of data is maximized. Thus we can define PCA in a recursive way. Suppose we are given a m-dimensional random variable $\underline{X}$, such that $\mathrm{E}\{\underline{X}\} = \underline{0}$, which means that $\underline{X}$ is centered. Certainly, this condition can be easily satisfied because we can subtract the expected value from the random variable $\underline{X}$. The first principle component $\underline{w}_1$ can be defined as:

$$\underline{w}_1 = \arg\max_{\|\underline{w}\|=1} \mathrm{E}\{(\underline{w}^T \underline{X})^2\}. \tag{5.1}$$

Thus the first principal component is the projection on the direction in which the variance of the projection is maximized. The successive components are defined as principle components of the remaining data. This results in the following recursive expression:

$$\underline{w}_k = \arg\max_{\|\underline{w}\|=1} \mathrm{E}\left\{\left(\underline{w}^T\left(\underline{X} - \sum_{i=1}^{k-1} \underline{w}_i \underline{w}_i^T \underline{X}\right)\right)^2\right\}. \tag{5.2}$$

Afterwards, the amount of the ith component in data is given by $s_i = \underline{w}_i^T \underline{X}$. The example data cloud and principle components are depicted in Figure 5.1.

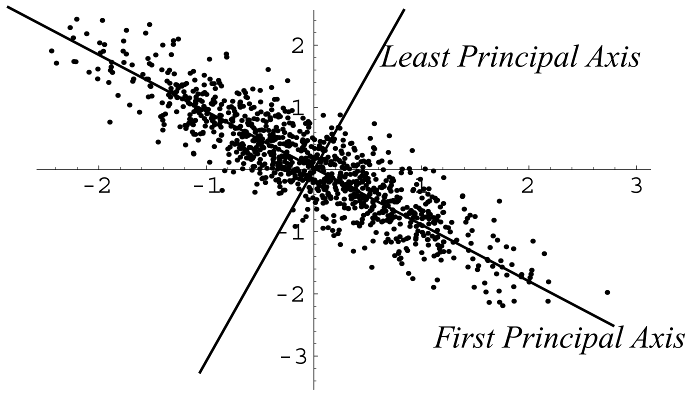

Figure 5.1: The principal components of an example two dimensional data cloud.

The above recurrent formula is used in lateral orthogonalization neural networks [38], where the neurons of in the hidden layer compute principle components. Assume that the first component is already obtained by the first neuron in the hidden layer. Then we can deflate the input of the second neuron by the amount of principle component determined in the input pattern: $\underline{\tilde{x}} = (I - \underline{w}_1 \underline{w}_1^T)\underline{x}$ and then repeat this process until the desired number of principle components is determined. By this manner, the recursive definition of principle components, given by Equations 5.1 and 5.1, is effectively implemented in an adaptive manner.

In practise, the computation of the $\underline{w}_i$ can be simply accomplished using the (sample) covariance matrix $\mathrm{E}\{\underline{X}\underline{X}^T\} = C$. The $\underline{w}_i$ are the normalized eigenvectors of C that correspond to the n largest eigenvalues λ_i of C. Each eigenvalue λ_i gives the variance of data in the direction $\underline{w}_i$.

The reduction of input space is accomplished by choosing only components of highest values λ_i. It can be proven [33], that such representation of data is optimal in the sense of mean square error.

In Machine Learning such approach proves useful only on condition that necessary components of data have the highest variance, which results in the highest eigenvalues, whereas components of the lowest eigenvalues represent noise of data. However, it is not necessarily always true. Consider the following decision table and assume the uniform data distribution.

Table 5.1: Example decision table

x_1	x_2	x_3	y
4	8	0	1
2	0	1	0
4	7	0	1
2	2	1	0
12	6	1	0
18	8	0	1
18	10	1	0
9	7	0	1
2	1	1	0

After centering the data we get the following principle components and corresponding eigenvalues:

$$\begin{array}{ll} \lambda_1 = 42.31 & w_1 = (-0.963, -0.269, 0.012) \\ \lambda_2 = 5.62 & w_2 = (-0.269, 0.961, -0.065) \\ \lambda_3 = 0.22 & w_3 = (-0.006, 0.066, 0.998) \end{array}$$

According to Table 5.1, the most important variable is x_3, since it is directly correlated with the output variable y. However, the component w_3, which mostly indicates the direction of x_3 ($w_3^T \cdot (0, 0, 1) = 0.998$), is associated with the lowest eigenvalue, and could be probably removed as an insignificant information for the computation. It flows from the fact, that in PCA the output information is not taken into cosideration.

Moreover, PCA is capable of reducing only linear dependencies between inputs, since eigenvectors are orthogonal and thus uncorrelated. As it has been shown in Example 1, uncorrelated variables are not necessarily independent, and thus nonlinear dependencies are not removed by PCA.

5.2 Independent Component Analysis

Recently, an improved method of component analysis, namely Independent Component Analysis (ICA), gained a widespread attention. In comparison to PCA method, the estimated components tend to be statistically independent, which means, that the functional dependencies between them can in some cases be reduced. ICA is usually applied in blind source separation, when the observed values of $\underline{X}$, corresponding to realization of an m-dimensional discrete time signal, are a mixture of some independent signal. The situation when the sources of independent signals are (like people talking simultaneously) linearly mixed together is quite often (for example in physics, biology, engineering). The goal of ICA is to find a linear transform which allows to identify these independent signals.

Definition 23 *ICA of the random vector $\underline{X}$ consists in finding a linear transform $\underline{s} = W\underline{x}$ so that the components s_i are as independent as possible, in the sense of maximizing some function $F(s_1, \ldots, x_m)$, which measures independence.*

The independence is most often defined by means of maximizing non-Gaussianity of data, since, according to central limit theorem, the distribution of a mixture of non-Gaussian signals tends to have the Gaussian distribution. The non-Gaussianity is often measured in terms of entropy and mutual information (as defined in Chapter 2), because the Gaussian distribution maximizes the entropy of a random variable of a fixed variance.

Although this method of redundancy removing proves very useful in signal analysis, for example in medicine [86], financial data analysis [36], telecommunications [63], it cannot be directly applied to the problem of argument reduction of input data, since the independent signals are not ordered in any way, and there is no clear indication which signals are needed for further processing. In the above mentioned applications, the researcher, basing on his prior knowledge, is responsible for choosing the right signals for computing.

5.3 Argument reduction

The notion of argument reduction introduced by Pawlak in [55] with regard to information systems certainly remains valid in the area of neural networks. As it has been already mentioned, quite often neural network has inputs which do not carry important information, however, they need to be processed, which results in larger neural networks. The approach capable of identifying such inputs will be presented in this section.

5.3.1 Introduction

Suppose we are given a truth table describing a multi-valued function computed by the network. An input variable x_k is called *dispensable* when for all pairs of input vectors from this decision table that differ only at variable k, the values of all output variables of the function are the same. A dispensable variable can be removed from the decision table without affecting the output values.

Example 5 *Examine the function given by Table 5.2. In this decision table only variable x_1 is dispensable, because in case of its removal the output values are still compatible. Whereas, when we try to remove x_2, we find the outputs contradictory when $x_1 = 2$ and $x_3 = 1$. Table 5.3 shows the function after argument reduction.*

Table 5.2: Example decision table before reduction.

x_1	x_2	x_3	y_1	y_2
3	1	1	0	3
2	1	1	0	3
2	2	1	1	0
2	2	2	2	2
1	0	3	1	2

Table 5.3: Example decision table after reduction.

x_2	x_3	y_1	y_2
1	1	0	3
2	1	1	0
2	2	2	2
0	3	1	2

From the above example we clearly see that certain variables discriminate non-compatible outputs. This leads us to the fundamental notion of discernibility matrix which for each pair of rows of the decision table determines the input variables necessary to discriminate incompatible outputs.

Let n be a number of rows in the decision table. A *discernibility matrix* is a matrix $\{m_{i,j}\}$, $n \times n$, where $m_{i,j}$ is a set of input variables needed to discriminate the output variables in ith and jth row. More formally speaking,

$$m_{i,j} = \begin{cases} \{k : x_k(i) \neq x_k(j)\} & \text{iff} \quad \exists l, y_l(i) \neq y_l(j) \\ \emptyset & \text{otherwise} \end{cases},$$

where $x_k(i)$ denotes the value of kth input variable in ith row, and $y_l(i)$ indicates the value of lth output variable in ith row. Such matrix can be computed in $O(n^2p)$ steps, where p denotes the total number of input and output variables.

Table 5.4: Example discernibility matrix.

	1	2	3	4	5
1	$\emptyset$	$\emptyset$	$\{1,2\}$	$\{1,2,3\}$	$\{1,2,3\}$
2	$\emptyset$	$\emptyset$	$\{2\}$	$\{2,3\}$	$\{1,2,3\}$
3	$\{1,2\}$	$\{2\}$	$\emptyset$	$\{3\}$	$\{1,2,3\}$
4	$\{1,2,3\}$	$\{2,3\}$	$\{3\}$	$\emptyset$	$\{1,2,3\}$
5	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\emptyset$

A discernibility matrix for function shown in Table 5.2 is given in Table 5.4.

There arises the problem of finding the maximal set of dispensable variables that can be removed. This task is called *argument reduction problem*.

Having determined the discernibility matrix, the argument reduction problem is nothing else but finding a minimum set of variables that has at least one common element with all nonempty sets in the matrix. This is equivalent to *minimum set cover* of a hypergraph.

Definition 24 *A* hypergrapf *is a pair of sets (V, E), where V is a set of* vertices *and E is a set of* edges. *Each edge connects an arbitrary number of vertices, $E \subseteq 2^V$.*

Definition 25 *Given a hypergraph $G = (V, E)$, a* set cover *of this hypergraph is any subset $C \subseteq V$ such that $\forall e \in E, e \cap C \neq \emptyset$. The cardinality $|C_{min}|$ of the* minimum set cover *C_{min} in a given hypergraph G is denoted by $\tau(G)$.*

Now the set of input variables corresponds to the set of vertices V, and the sets of variables in the discernibility matrix are the edges of the hypergraph. Algorithm 4 describes the argument reduction with application of *minimum set cover* of a hypergraph.

Algorithm 4 Argument reduction.

```
Create a hypergraph $G$ with an empty set $E$ of edges and a set $V$ of vertices such
that each vertex corresponds to one input variable of the function;
for all pairs of rows in the decision table which have incompatible output values do
    Determine the input variables of different values in these rows;
    Add an edge, if such does not exist, joining the vertices corresponding to these
    input variables;
end for
Find the minimum set cover in $G$;
Remove from the decision table the input variables whose corresponding vertices are
not present in minimum set cover and reconstruct the table;
Train the neural network using the reduced decision table;
```

Example 6 *Consider once again the function defined in Table 5.2. Algorithm 4 constructs a hypergraph $G = (V, E)$ such that $V = \{1, 2, 3\}$, $E = \{\{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$. The hypergraph G is depicted in Figure 5.2. A minimum set cover algorithm finds $C_{min} = \{2, 3\}$ and a new decision table is created — Table 5.3.*

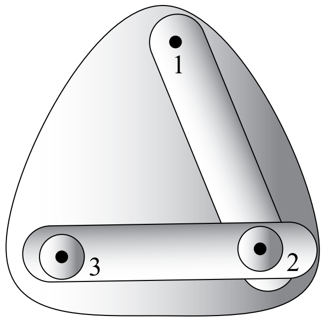

Figure 5.2: A hypergraph corresponding to the decision table given in Table 5.2.

5.3.2 Minimum set cover algorithms

First of all the exact algorithm for minimum set cover will be presented. The variable reduction is performed in a backtracking way. Successive vertices are removed from the hypergraph as long as the vertex cover is maintained. Otherwise, the algorithm goes back one level in backtracking tree. Each time the best solution is saved. The whole procedure is given in Algorithm 5. The vertices are successively numbered starting from 1.

Algorithm 5 Backtracking algorithm for minimum set cover.

```
Require: $C$ – the set of vertices, $v$ – the first vertex to remove from $C$; in the beginning
  $C = V$ and $v = 1$;
  for all $w \in C$ such that $w \geq v$ do
    if $C - \{w\}$ is a set cover and $C - \{w\}$ was not already visited then
      Set $C_{min} := C$ if $C$ has less elements than $C_{min}$ (the current best solution);
      Call the procedure recursively with parameters $C := C - \{w\}$, $v := w$;
    end if
  end for
```

Unfortunately, Algorithm 5 is NP-hard and cannot be applied for larger decision tables. It is necessary to apply a heuristic — a *Lovász-Johnson-Chvatal algoritm* [18] given as Algorithm 6.

Certainly Algorithm 6 is not optimal. The cardinality of minimum set cover found by it is $\tau'(H) \leq \tau(H)(1 + \log |V|)$, where $\tau(H)$ denotes the optimal solution. Consider

Algorithm 6 Greedy algorithm for minimum set cover.

```
Require: $V$ – the set of vertices, $E$ – the set of edges;
  $S_{min} = \emptyset$;
  while $E$ is not empty; do
    $v :=$ the vertex that covers most edges;
    $S_{min} := S_{min} \cup \{v\}$;
    $V = V \setminus \{v\}$;
    $E := E \setminus \{e : v \in e\}$;
  end while
```

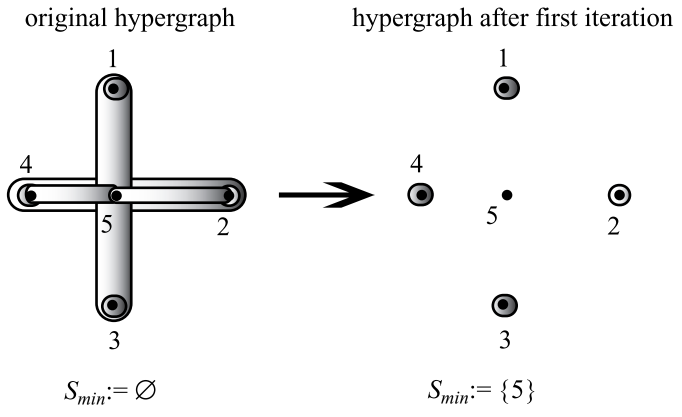

Figure 5.3: First iteration of Algorithm 6.

the example of its behaviour depicted in Figure 5.3. Vertex 5 covers the largest number of edges and thus is chosen in the beginning. Unfortunately, the remaining vertices also should be included in the set cover. Finally, although the minimum set cover is $\{1, 2, 3, 4\}$, all the vertices $\{1, 2, 3, 4, 5\}$ are chosen.

The performance of Algorithm 6 can be improved, when, before its application, a simple preprocessing is done. Let us consider simple properties of minimum set cover cover.

- Suppose an edge e_1 is included in an edge e_2, then if e_1 is covered e_2 is also covered and thus e_1 can be removed from the hypergraph.
- Let v_1 and v_2 be such vertices that $\forall e \ni v_2,\ v_1 \in e$. It is denoted as $v_1 \supset v_2$. All edges covered by v_2 are also covered by v_1. Therefore there is no need to consider v_2 when constructing minimal set cover and v_2 can be removed.
- If an edge is a singleton, the vertex corresponding to it must be present in a minimum set cover.

The above remarks are implemented in the hypergraph preprocessing procedure given by Algorithm 7.

Algorithm 7 Greedy algorithm for minimum set cover.

```
for all $e_1 \in E$ such that $\exists e_2 \subset e_1$ do
    Remove $e_1$ from the hypergraph;
end for
for all $v_1 \in V$ such that $\exists v_2 \supset v_1$ do
    Remove $v_1$ from the hypergraph;
end for
for all $e \in E$ such that $e = \{v\}$ do
    $C_{min} := C_{min} \cup \{v\}$;
    Remove $v$ and $e$ from the hypergraph;
end for
```

Again consider the hypergraph in Figure 5.3. Figure 5.4 depicts preprocessing. Observe, that after application of Algorithm 7, the greedy set cover algorithm achieves better results.

5.3.3 Argument reduction and entropy

Suppose we are given a decision table F and a certain probability distribution on its input variables $x_1, x_2, \ldots, x_n$. It means that each row i of a decision table has a corresponding probability p_i of occurrence. Therefore we can treat the input variables of a function given by the decision table as a multidimensional random variable X and we

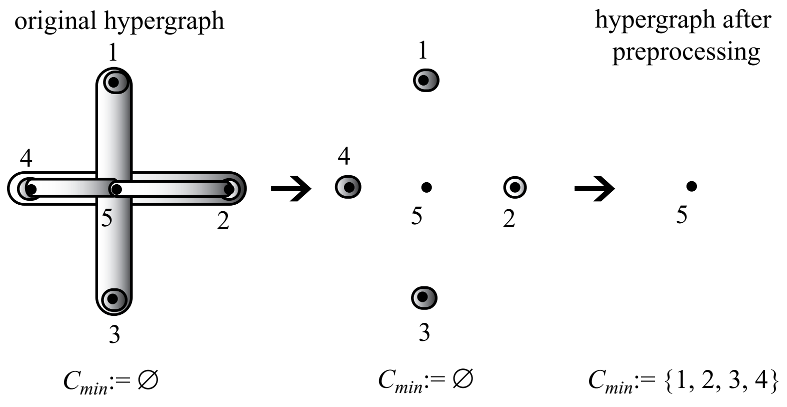

Figure 5.4: Preprocessing of an example hypergraph.

can determine, in term of entropy $\mathrm{H}(X)$ (see Chapter 2), the amount of information connected with it. We have

$$\mathrm{H}(X) = \sum_{i=1}^{m} -p_i \log(p_i),$$

where m denotes the number of rows in the decision table.

After argument reduction some input variables are removed. After that, as it has been discussed in Subsection 5.3.1, compatible rows are unified. Let m' denote the number of rows in the decision table F' after reduction. The probability corresponding to a unified row is a sum of the probabilities, corresponding to each of rows before unification. Assume that function $U(i')$ returns a set of rows of the decision table F which have been unified. We define $a = U^{-1}(i) \Leftrightarrow i \in U(a)$. The entropy $\mathrm{H}(X')$, where X' is an input random variable after argument reduction, is therefore as follows:

$$\begin{aligned}
\mathrm{H}(X') &= -\sum_{i'=1}^{m'} p'_{i'} \log(p'_{i'}) = -\sum_{i'=1}^{m'} \left(\sum_{i \in U(i')} p_i \right) \cdot \log \left(\sum_{j \in U(i')} p_j \right) \\
&= -\sum_{i'=1}^{m'} \sum_{i \in U(i')} p_i \cdot \log \left(\sum_{j \in U(i')} p_j \right) = -\sum_{i=1}^{m} p_i \cdot \log \left(\sum_{j \in U(U^{-1}(i))} p_j \right) \\
&\leq -\sum_{i=1}^{m} p_i \cdot \log(p_i) = \mathrm{H}(X), \qquad (5.3)
\end{aligned}$$

with equality if no variable have been removed. In the above equation the following property has been used: $\log(p_i + p_j) \geq log(p_i)$, $p_i, p_j > 0$.

According to Equation 5.3, the argument reduction reduces also the entropy of input variables, however, the information necessary to compute the output variables remains.

Chapter 6

Decomposition of neural networks

Generally speaking, decomposition appears as a fundamental problem of great importance for wide spectrum of different scientific fields. The strong motivation for developing decomposition techniques comes from modern research areas such as pattern recognition, knowledge discovery and machine learning. It is also useful in logic synthesis, in computer-aided design of very large integrated circuits. Simply speaking, decomposition means dividing a complex problem into several relatively easier and independent sub-problems. In this way we can divide one complex neural network into some smaller and simpler sub-networks and thus achieve better convergency and shorter time of training. Thus decomposition appears as a powerful tool in neural network design.

In gradient decent methods (see Chapter 3) of supervised training, the training set consists of pairs input-output. It is nothing else but a decision table. The goal is to find the structure of a neural network suitable for given data, which is a difficult problem in general. Although, some research in this field has been done, there is currently a lack of a general method. It turns out that functional decomposition helps to structure the neural network according to the given data.

6.1 Problem formulation

Suppose we intend to model a behaviour of a certain information system with application of a feedforward neural network. As it has been already described in Chapters 2 and 3, the supervised training of such model is carried out on the basis on pairs giving the output of the system in correspondence to the input. The size of the network obviously depends on the amount of data it has to process (see Chapter 4). The serial decomposition allows to distribute processing among several blocks. Consider again the process of training the neural network. Notice, that feedforward neural networks are usually trained in an off-line manner, i.e. the input-output pairs are collected as a training set. These sets are often multi-valued and thus may be considered as a multi-valued decision tables. Fortunately, the algorithms originally invented to structure digital circuits, have been generalized and prove useful in decomposition of such decision tables.

Since all the information that can be extracted by a neural network is given by a decision table (a training set), the decomposition of the decision table corresponds to the decomposition of the neural network. Therefore the procedure of neural network decomposition can be expressed as follows:

1. Suppose we are given a multi-valued training set T of input-output pairs, which is not contradictory. It defines a function $F(X) = Y$, where X denotes a possibly multidimensional input value and Y is a corresponding multidimensional output of the network.

2. Decompose the function F in such a way that $F(X) = H(G(A), B)$, where $X = (A, B)$, G and H are certain functions with corresponding decision tables.

3. Device two neural networks corresponding to the decision tables G and H and use G and H as training sets.

4. Join the separate neural networks G and H.

An example feedforward neural network decomposition is depicted in Figure 6.1

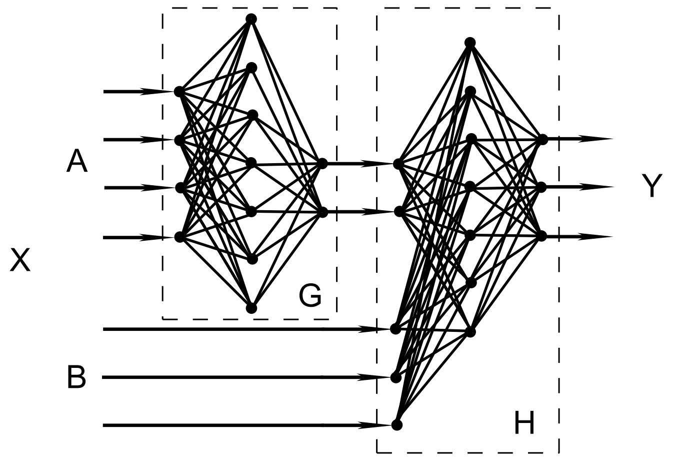

Figure 6.1: Neural network serial decomposition.

The decomposition method capable of finding G and H such that $F(X) = H(G(A), B)$ will be introduced in the next section.

6.2 Generalized Ashenhurst-Curtis decomposition

The notion of decomposition has been originally suggested by Ashenhurst and Curtis [5, 19]. Most of the work in this field has been done by professor Łuba from Warsaw University of Technology [41]. I will begin with the introduction of basic notions.

Let C_i and D be finite sets for all $i \in \{1, 2, \ldots, n\}$. The multiple-valued function F is defined as a mapping $F(x_1, \ldots, x_n) : C_1 \times C_2 \times \ldots \times C_n \rightarrow D$. Every element of the domain $C_1 \times C_2 \times \ldots \times C_n \rightarrow D$ is called a *minterm* (sample or object). A listing of minterms with the value of the function is called a *decision table* (or truth table in binary case). Decision tables do not include minterms for which the function values are not specified. For the sake of clarity this unspecified set of minterms will be called Don't Care Set (DC). There, the functions with non-empty DC-set will be called the partial functions.

Let F be a multiple-valued function representing functional dependency $Y = F(X)$, where $X = (x_1, x_2, \ldots, x_n)$ is the set of input variables and $Y = (y_1, y_2, \ldots, y_n)$ is the set of output variables. Suppose we are given the partition of input variables into two disjoint sets: a *bound set* A and a free set B. We say that there is a functional decomposition of F if $F(X) = H(G(A), B)$. More precisely speaking, G and H denote functional dependencies $Y = H(Z, A)$ and $Z = G(A)$. Function $H(G(A), B)$ is called a simple disjoint decomposition of function F. The structure of the decomposed decision function is shown in Figure 6.2. It is necessary to emphasize that the function after decomposition is equivalent to one before decomposition, since decomposition is a method which helps to structure a decision table.

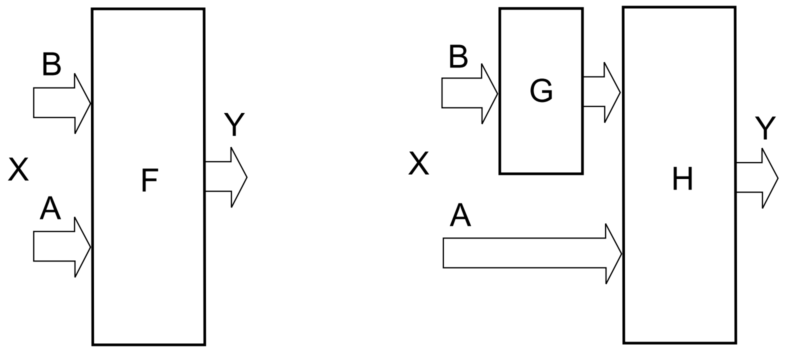

Figure 6.2: Serial decomposition scheme.

The interpretation of the scheme presented in Figure 6.2 is as follows. At the beginning, the intermediate decision is taken on the basis of attributes from subset A. Then consequently, the final decision is taken with respect to both intermediate decision $G(A)$ and attributes from partition B. The fundamental problem is then, given a function F and the partition of the set of input variables into disjoint sets A and B, to find the functions G and H, such that $F = H(G(A), B)$. To solve this problem, let us first introduce some useful concepts.

The *decomposition chart* of the function F is a two dimensional matrix with columns indexed using the values of the bound set variables A and rows indexed using the values of the free set B. Elements of the matrix $m_{i,j}$ are the values assumed by the function F for the input vectors $X = (A, B)$ constructed from ith row and jth column. We define a *column multiplicity* of the decomposition chart as the number of columns which cannot be unified. The column multiplicity of the matrix is denoted by $v(B|A)$.

Theorem 20 (Łuba) *Given a k-valued, m-output incompletely specified function F and a partition of its input variables into disjoint sets A and B, the serial decomposition in the form $F = H(G(A), B)$, where G is a l-valued, p-output function, exists if and only if there exists an expansion of the function F, for which $v(B|A) \leq l^p$.*

Proof See [40].

In Theorem 20 the term of expansion simply means the process of assigning arbitrary values to those combinations of inputs, for which the output is not specified. The goal is to find such such expansion that column multiplicity $v(B|A)$ is minimized.

6.3 Combinatorial approach

Before we formulate the algorithm it is necessary to introduce the concept of *compatible columns* in the decomposition chart. Two columns i and j in the decomposition chart are called *compatible* if for all rows in the chart, the corresponding elements in both columns are the same or at least one of them is "don't care". Certainly, the "don't care" element is compatible with any value. Columns which are not compatible are called *incompatible.* Theorem 20 suggests that the fundamental problem of decomposition is nothing else but finding an expansion of the function for which $v(B|A)$ has the least value, and this problem can be reduced to a graph coloring problem [72, 88]. Its objective is to color all the vertices of the graph with minimal number of colors, in such a way, that no two adjacent vertices (in our case columns, that are contradictory), are assigned the same color. Necessary definitions are given below.

Definition 26 *A* graph *is a pair of sets (V, E), where V is a set of vertices, and a set of edges E is a binary relation defined on V, $E \subseteq V \times V$. The graph is* undirected, *when relation E is symmetric: $\forall u, v \in V,\ (u, v) \in E \Rightarrow (v, u) \in E$. (In this work only undirected graphs are considered.) The vertices $u, v \in V$ are called adjacent if $(u, v) \in E$. The number of vertices adjacent to a given vertex v is called the* degree *of v and denoted as $d(v)$.*

Definition 27 *A k-*coloring *of a given graph G is a function $c : V \to C$, such that no adjacent vertices are assigned the same color. C is a set of colors of cardinality k. A graph is said to be k-*colorable *if there exists a k-coloring of this graph. The minimal number k for which a given graph is k-colorable is a* chromatic number *$\chi(G)$.*

Unfortunately, the minimum graph coloring algorithm is NP-hard, and usually only suboptimal solutions are achieved when heuristic approaches are applied.

The combinatorial approach to the decomposition is expressed as Algorithm 8.

Algorithm 8 Graph coloring based decomposition.

```
Require: Function $F$ to be decomposed, a partition of input variables int subsets $A$ and $B$;
  (1) Construct a graph $G_F$ in which each vertex correspond to a column in the decomposition chart and two vertices are connected with an edge if the corresponding columns are incompatible;
  (2) Color the graph $G_F$;
  (3) Construct the functions $G$ and $H$ basing on the coloring of $G_F$;
```

Step (1) and (3) of Algorithm 8 can be done in polynomial time with respect to m and n, where m is a number of columns and n is the number of rows in the decomposition chart. Step (2) must be carried out in a heuristic way, since the exact algorithm for minimum graph coloring is NP-hard.

6.3.1 Example

Let us study an example to illustrate a run of Algorithm 8.

Example 7 *Consider a 3-valued function presented in Table 6.1.*

Table 6.1: Decision table of function F.

x_1	x_2	x_3	x_4	x_5	y_1	y_2
0	0	0	0	0	0	2
2	0	2	2	0	2	1
1	1	1	0	0	0	2
1	1	1	1	1	1	0
0	0	0	1	1	1	0
2	2	2	2	0	2	1
1	1	2	1	1	1	3
1	0	1	0	0	1	1
1	1	1	2	0	1	2
1	1	2	0	0	2	2

This function has 5 input and 2 output variables. Suppose the input variables are partitioned as follows: $A = (x_1, x_2, x_3)$, $B = (x_4, x_5)$. The decomposition chart is given in Table 6.2.

Now, using the decomposition chart, we can construct the incompatibility graph which is given in Figure 6.3.

Afterward graph coloring is conducted. The vertices are assigned the following colors:

$$c(0,0,0) = 0, \quad c(1,0,1) = 1, \quad c(111) = 0,$$
$$c(1,1,2) = 2, \quad c(2,0,2) = 1, \quad c(222) = 1.$$

Table 6.2: Decomposition chart for function F

	$x_1x_2x_3$					
x_4x_5	000	101	111	112	202	222
00	02	11	02	22	-	-
11	10	-	10	13	-	-
20	-	-	12	-	21	21

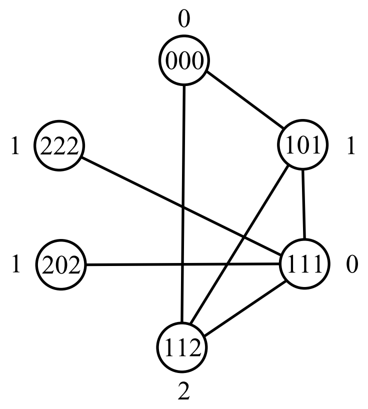

Figure 6.3: Incompatibility graph constructed from the decomposition chart given in Table 6.2 and corresponding graph coloring.

Since only 3 colors have been used and the decomposed function is 3-valued, the intermediate function G is constructed with only one output variable. Now the colors are coded as 3-valued numbers. In this example an identity mapping can be applied. Finally, in conformity with the chosen coding, decision tables of functions G and H are constructed and they are given in Tables 6.3 and 6.4.

Table 6.3: Subfunction G after decomposition.

x_1	x_2	x_3	g
0	0	0	0
1	0	1	1
1	1	1	0
1	1	2	2
2	0	2	1
2	2	2	1

Table 6.4: Subfunction H after decomposition.

x_2	x_3	g	h_1	h_2
0	0	0	0	2
0	0	1	1	1
0	0	2	2	2
1	1	0	1	0
1	1	2	1	3
2	0	0	1	2
2	0	1	2	1

Note that we achieved a serial decomposition of function F, since, as it can be directly verified, $F(X) = H(G(A), B)$.

6.4 Graph coloring heuristics

As it has been already mentioned, the minimum graph coloring problem is NP-hard. An optimal approach, namely the backtracking algorithm, cannot be applied to larger graphs. Therefore, to achieve a decomposition by means of Algorithm 8, the application of heuristic graph coloring is inevitable.

6.4.1 The family of sequential algorithms

A sequential algorithm bases on a greedy approach. Given an ordering of vertices $(v_0, v_2, \ldots, v_{n-1})$, in each step, it assigns a lowest possible color to a consecutive vertex.

The procedure of sequential graph coloring is given as Algorithm 9. The colors are denoted with nonnegative integers. The output variable k gives the number of colors used in coloring.

Algorithm 9 Sequential graph coloring algorithm.

```
Require: $(v_0, v_1, \ldots, v_{n-1})$ — the given ordering of vertices in the graph;
  $k := 1$;
  for $i := 0$ to $n-1$ do
    Color $v_i$ with least possible color $c$;
    if $c = k$ then
      $k := k + 1$;
    end if
  end for
```

Algorithm 9 is fast and simple, however its results strongly depend on vertex ordering. Consider a planar graph of $2n$ vertices and the following ordering of vertices: $(v_0, v_1, \ldots, v_{n-1})$ (Figure 6.4). Although graph G is a bipartite graph and thus can be colored with 2 colors, as much as n colors have been used when applying the sequential algorithm.

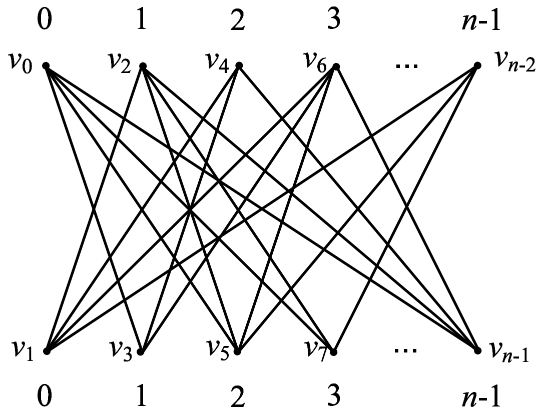

Figure 6.4: A bipartite graph colored with sequential algorithm.

Numerous attempts have been taken to improve the behaviour of the sequential algorithm by means of supporting an appropriate vertex ordering. Two approaches are especially worth being mentioned:

LF In Largest First Sequential Algorithm the vertices are non-increasingly ordered with respect to the degree of the vertices.

SL The ordering in the Smallest Last Sequential Algorithm is obtained in the following way. The vertex of the lowest degree in a graph G is added to the vertex ordering list as a first element and is then removed from the graph. This step is repeated until the graph is empty.

6.4.2 Maximum independent set algorithm

As we have observed, the sequential algorithm can behave in an very poor way, and moreover, there is no upper bound relating its solution to the optimal one. Maximum independent set algorithm is an approach that guarantees that no more than $3|V|/\log_k |V|$ colors are used, where $k = \chi(G)$ and $G = (V, E)$. It is based on the fact that the vertices from an *independent set* W (no two vertices in such set are adjacent) can be assigned the same color. In each step the algorithm finds the *maximum independent set* and assigns to its vertices the same consecutive color and afterwards removes them with corresponding edges form the graph. The procedure is repeated until the graph is empty.

Unfortunately, the maximum independent set problem is NP-hard and a heuristic method has to be applied. The approximate independent set W is found in an iterative manner. A vertex of the lowest cardinality v is is added to W. Afterwards v is removed with its neighbours (vertices that are adjacent to v) from the graph. The step is iterated while the graph is not empty.

The above discussed approach to minimum graph coloring problem is expressed as Algorithm 10.

Algorithm 10 Approximate maximum independent set algorithm.

```
Require: Graph $V$ to be colored;
  $i := 1$, $U := V$;
  while $U \neq \emptyset$ do
    W:=U;
    while $W \neq \emptyset$ do
      $H$ :=subgraph induced in $G$ by $H$;
      $v$ :=vertex of minimum degree in $H$;
      Color $v$ with the color i;
      $W := W - \{v\} - \{u : u$ is a neighbour of $v$ in $H\}$;
      $U := U - \{v\}$;
    end while
    $i := i + 1$;
  end while
```

6.5 Multi-block serial decomposition

The decomposition method introduced in Section 6.2 allows to divide a monolithic system into two blocks. However, it can be easily generalized so that multi-block structures can be achieved. If a function F is decomposed in a classical way: $F(X) := H(G(A), B)$, where $X := (A, B)$, function G may be decomposed in such way, that the output of $G(A)$ constitutes a free set and B is a bound set. Hence, we get the following decomposition: $F(X) := I(G(A), J(B))$. The situation is depicted in Figure 6.5. In similar way, more complicated structures can be achieved.

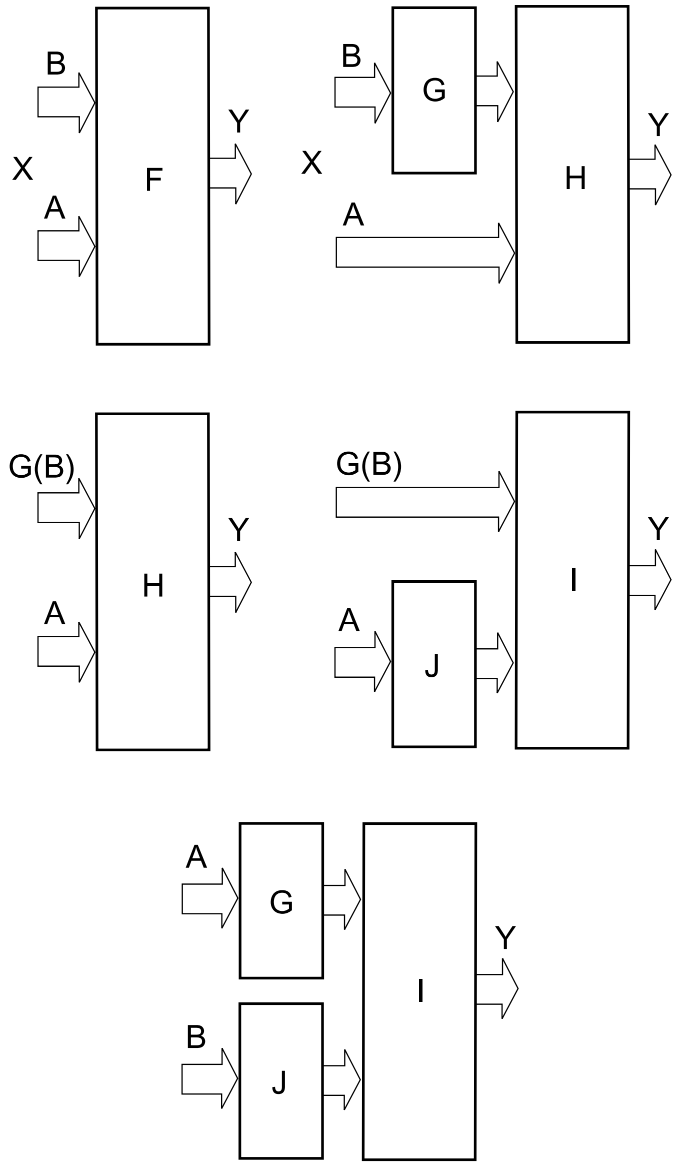

Figure 6.5: A multi-block serial decomposition.

6.6 Variable ordering

In Section 6.2 it has been assumed that we are given a partition of input variables into a bound set A and a free set B. It turns out, that the quality of decomposition strongly depends on proper partitioning. Some methods like exhaustive search (only in case of small functions), genetical programming and another heuristic approaches can be applied, however, there is a need for a more general method. In [17] we suggest a method of finding a variable partitioning from the structure of a Reduced Ordered Multi-valued Decision Diagram (ROMDD). This approach is based on a well-known *sifting* algorithm. In our procedure, the objective is to find such an ordering of variables in ROMDD, so that the width of the diagram is minimized at a given level. The variables above this level constitute a bound set A, and those below — form a free set B.

The details of the method and experimental results can be found in [17].

6.7 Decomposition and entropy

At the very beginning of this work I depicted the process of learning of an arbitrary information system in terms of entropy. We have already seen that argument reduction also reduces entropy. And what are the entropies of inputs of blocks G and H in case of a serial decomposition $F(X) = H(G(A), B)$?

Again, as it was done when argument reduction was considered, assume that we are given a decision table F of m rows and a certain probability distribution on its input variables $x_1, x_2, \ldots, x_n$. Thus p_i denotes the probability of occurrence of ith row.

Now consider the decomposition chart after unifying the compatible columns. Let $j = 1, \ldots, n$ be an index of columns and $k = 1, \ldots, o$ be an index of rows. We define also a function $U(j,k)$ which returns a set of indexes $\{i\}$ of rows from decision table F, which have been unified. The probabilities corresponding to unified columns are the sums of respective probabilities. The entropy of $Z = G(A)$ is therefore

$$\begin{aligned}
\mathrm{H}(Z) &= -\sum_{j=1}^{n} \left(\sum_{k=1}^{o} \sum_{l \in U(j,k)} p_l \right) \log \left(\sum_{k=1}^{o} \sum_{l \in U(j,k)} p_l \right) \\
&= -\sum_{i=1}^{m} p_i \log \left(\sum_{k=1}^{o} \sum_{l \in U(j,k): \exists r,\ i \in U(j,r)} p_l \right) \\
&\leq -\sum_{i=1}^{m} p_i \log(p_i) = \mathrm{H}(X), \qquad (6.1)
\end{aligned}$$

with equality if no columns have been joined in the decomposition chart.

In similar way we compute the entropy $\mathrm{H}(G(A), B)$ of the input of the block H:

$$\begin{aligned}
\mathrm{H}(G(A), B) &= -\sum_{j=1}^{n}\sum_{k=1}^{o}\left(\sum_{l\in U(j,k)}\right)\log\left(\sum_{l\in U(j,k)}\right) \\
&= -\sum_{i=1}^{m} p_i \log\left(\sum_{l\in U(j,k):i\in U(j,k)} p_l\right) \\
&\leq -\sum_{i=1}^{m} p_i \log(p_i) = \mathrm{H}(X), \qquad (6.2)
\end{aligned}$$

with equality if no columns in the decomposition chart have been unified.

According to the above inequalities, the entropy of inputs of both blocks is lower that the entropy of X. Therefore the blocks G and H (and corresponding neural networks) have less amount of information to process and can be of smaller size.

Chapter 7

Experimental results

The idea of decomposition of neural networks has been verified using real life examples. Numerous tests were performed to study the process of learning of decomposed neural networks. Generally speaking, various benchmarks were decomposed and then a neural network was created for each block. Afterwards the training process was examined. The computer experiments were conducted on a 2×Pentium III 800 MHz machine running GNU/Linux operating system. The tests were performed mostly for a set of standard benchmarks from Collaborative Benchmarking Laboratory.

7.1 Tools

7.1.1 Stuttgart Neural Network Simulator

Network simulations were conducted using Stuttgart Neural Network Simulator (SNNS) [76]. SNNS is a software simulator for neural networks on Unix workstations developed at the Institute for Parallel and Distributed High Performance Systems (IPVR) at the University of Stuttgart. SNNS had been chosen for simulation because of its efficiency and flexibility. It is a well-established simulation environment for research on and applications of neural networks. It offers a variety of architectures, algorithms and other features pertinent to neural networks. It also allows batch processing which considerably increases the comfort of work. The effects and progress of training can be observed in graphical interface.

7.1.2 HOSEA

HOSEA [70, 71] is a software developed by Michał Pleban at the Institute of Telecommunications at Warsaw University of Techology. It implements the algorithms described in Chapters 5 and 6, and is capable of performing the following tasks:

Serial decomposition From one input function it generates two new functions corresponding to G and H blocks in the decomposition scheme.

Parallel decomposition It generates two new functions, corresponding to the two subsets of output variables.

Argument reduction The application reduces the number of arguments of the given function, and generates new function with reduced set of inputs.

7.2 Results of experiments

Feedforward neural networks were used in the experiments. The neural network training was carried out with application of standard backpropagation algorithm and its modifications, mainly Scaled Conjugate Gradient algorithm. Unfortunately, the architectures of neural networks cannot be defined in an automatic way. Thus, crucial parameters of networks, namely a number of hidden layers and numbers of neurons in each hidden layer, were chosen manually in an iterative way, so that the performance of training was maximized. All neurons were equipped with standard logistic activation function. Weights associated with interneuronal connections were initialized with small random numbers. Most of the results given below were published in [53, 42, 71, 72].

7.2.1 Small and medium benchmarks

First of all, the decompositions of small and medium benchmarks were examined. Tables 7.1 and 7.1 show the results. Each benchmark was decomposed into two blocks G and H of possibly similar number of inputs. For each block a neural network was designed and trained separately. Afterwards, the networks were joined together to constitute a modular neural network. The column "block" denote the trained block, where F stands for a network without decomposition. For each block number of input variables "inp. var. #" and output variables "out. var. #" is given. Each benchmark was trained several times. Tables 7.1 and 7.1 give three examples of training of each benchmark for various network architectures. The column "train time" gives the time of training in seconds. Number of erroneous patterns for each module is indicated in the column "err. #".

These results clearly testify, that decomposition of neural networks is a good approach to achieve the benefits of modularity. Almost all decomposed examples were trained in a shorter time than the benchmarks without decomposition, and in many cases decomposition allowed to avoid errors.

7.2.2 Large neural networks

The data used in this part of experiments were acquired as a result of air pollution measurements of a power plant. The dependencies between the operating parameters and air pollution are given by a 4-valued decision table of 60 inputs, 24 outputs and 17688 disjoint patterns. The structure of the data used in process of training is described below.

Table 7.1: Training results for small and medium benchmarks

benchmark name	block	inp. var.#	out. var.#	train. time	err. #	train. time	err. #	train. time	err. #
9SYM	F	9	1	232	0	131	0	600	57
	G	5	3	2	0	2	0	5	0
	H	7	1	2	0	6	0	18	0
ADR4	F	8	5	179	0	268	0	433	0
	G	4	3	1	0	1	0	1	0
	H	7	5	49	0	47	0	6	0
BBTAS	F	5	5	600	1	5	0	1	0
	G	3	3	1	0	1	0	1	0
	H	5	5	1	0	13	0	1	0
BEECT	F	6	7	111	0	601	2	228	0
	G	3	3	1	0	1	0	1	0
	H	6	7	5	0	5	0	4	0
CLIP	F	9	5	602	181	601	187	603	541
	G	5	4	5	0	4	0	30	0
	H	8	5	214	0	330	0	533	0
EX7	F	6	6	59	0	11	0	9	0
	G	5	4	1	0	20	0	2	0
	H	5	6	3	0	33	0	1	0
MISEX	F	8	7	58	0	65	0	81	0
	G	4	3	1	0	1	0	1	0
	H	7	7	24	0	23	0	27	0
OPUS	F	9	10	608	152	600	117	604	6
	G	5	4	37	0	3	0	1	0
	H	8	10	90	0	160	0	109	0
RD53	F	5	3	7	0	49	0	5	0
	G	3	2	1	0	1	0	1	0
	H	4	3	2	0	5	0	1	0
RD73	F	7	3	221	0	7	0	600	22
	G	5	3	2	0	1	0	31	0
	H	5	3	6	0	1	0	50	0
RD84	F	8	4	600	3	601	3	600	3
	G	5	3	7	0	44	0	5	0
	H	6	4	7	0	4	0	27	0
ROOT	F	8	5	169	0	234	0	95	0
	G	6	5	4	0	11	0	4	0
	H	7	5	5	0	16	0	6	0
S8	F	7	4	240	0	34	0	20	0
	G	4	3	1	0	1	0	1	0
	H	6	4	3	0	2	0	3	0

Table 7.2: Training results for small and medium benchmarks, continued

benchmark name	block	inp. var.#	out. var.#	train. time	err. #	train. time	err. #	train. time	err. #
SAO2	F	10	4	600	35	600	36	600	24
	G	7	4	219	0	601	5	600	2
	H	7	4	9	0	8	0	19	0
SQRT8	F	8	4	37	0	601	18	53	0
	G	4	2	1	0	1	0	1	0
	H	6	4	1	0	92	0	3	0
XOR5	F	5	1	600	2	600	2	600	3
	G	3	1	256	0	1	0	117	0
	H	3	1	1	0	1	0	51	0
Z4	F	7	4	13	0	88	0	18	0
	G	4	3	1	0	1	0	1	0
	H	6	4	1	0	13	0	14	0

Inputs of the neural network:

parameter name	no. of variables
oil pressure	6
amount of anti soot substance	6
oil temperature	6
power of the generator	6
for each of 24 oil burners: oxygen valve open	12
for each of 24 oil burners: gas valve open	12
for each of 24 oil burners: oil valve open	12

Outputs of the neural network:

parameter name	no. of variables
air opacity	6
concentration of NO_x	6
oil temperature	6
power of the generator	6
concentration of CO in eastern direction	6
concentration of CO in western direction	6

In order to find out how the results of the approach depend on the size of a pattern set, smaller training sets were prepared, using selection with uniform probability from original pattern set.

As I wanted to examine the influence of argument reduction and decomposition on the process of learning of neural networks, the training was carried out 4 times: with sets of not processed data, with decomposed data, with reduced data and with reduced and afterwards decomposed data.

The decision tables (and therefore neural networks) were decomposed into five subnetworks G_1, G_2, G_3, G_4, H connected in a serial way:

$$F(X) := H(G_1(A), G_2(B), G_3(C), G_4(D)), \tag{7.1}$$

where $X = (A, B, C, D)$. It is noteworthy, that there are many ways to design each subnetwork.

The application of argument reduction eliminated the redundant input variables from the blocks, what decreased the size of the training set.

Weights were initialized with random values. Different learning methods were applied: standard backpropagation, scaled conjugate gradient (SCG), resilient backpropagation (RPROP) and other. Generally, all the results were similar. The effects of training with SCG algorithm are presented in Tables 7.3 and 7.4. The time of training was limited to 3600 seconds. Certainly, if the training time had been extended the errors would have been reduced.

It is necessary to emphasize, that networks before and after decomposition are equivalent in information sense, as the decomposition methodology, what has been already stated, does not lose information. Therefore, the training error depends only on a training algorithm and computational capabilities of network induced by its structure.

It turned out (see Table 7.3), that it was impossible to train such huge neural network in conventional way, and in order to get smaller, manageable sub-networks, decomposition techniques described in the previous chapters had to be applied.

We can see that, after applying the decomposition, the mean square error as well as the number of erroneous patterns and bits decreased considerably. It is important to point out that before the application of the decomposition the data were too complex to be trained effectively. However, after the decomposition of data, we achieved a reasonable training results. As it is shown in Table 7.3, the most difficult problem was to properly learn the H sub-network. The reason for that is the difficulty of the intermediate decision encoding. I have found that the way in which the outputs of $G_i, i = 1 \ldots 4$, block are coded influences the convergence and the results of training.

Another interesting observation is the fact, that argument reduction decreased significantly the effectiveness of training. At first sight it seems to be counterintuitive, but it may be easily explained in terms of the theory introduced in Chapter 4, which, briefly speaking, defines the computational power of a neural network in relation to the number of its weights. Notice, that after argument reduction, the number of patterns in the training set remained almost the same (compare the first column of Tables 7.4 and 7.3). Thus, a computational power of a network should have remaind at the same level as in case of network without argument reduction. Unfortunately, the number of neurons in hidden layers was reduced and the network was unable to learn the training

Table 7.3: The training results (data without argument reduction, training time restricted to 3600 sec.)

	Original network			Decomposed network (into 5 blocks)			
# of patterns	Mean square error	# of error pat.	# of error bits	Sub-net's name	Mean square error	# of error pat.	# of error bits
1102	1.650	1102	11618	G_1	0.004469	23	23
				G_2	0.001245	4	4
				G_3	0.000510	0	0
				G_4	0.001531	1	1
				H	0.112735	119	154
2222	2.033	2222	26214	G_1	0.007715	66	66
				G_2	0.009632	71	71
				G_3	0.000812	0	0
				G_4	0.003609	3	4
				H	0.078617	100	128
4369	2.355	4369	82289	G_1	0.012796	167	180
				G_2	0.014283	183	196
				G_3	0.001735	0	0
				G_4	0.006161	3	5
				H	0.239675	310	715
8850	3.965	8850	153801	G_1	0.005193	6	6
				G_2	0.011377	11	15
				G_3	0.000640	0	0
				G_4	0.007335	8	12
				H	0.199367	284	532
17688	6.089	17688	312889	G_1	0.028686	636	716
				G_2	0.087505	1911	4431
				G_3	0.174261	20	20
				G_4	0.104712	82	87
				H	1.614805	2350	23892

Table 7.4: The training results (data after argument reduction, training time restricted to 3600 sec.)

	Original network			Decomposed network (into 5 blocks)			
# of patterns	Mean square error	# of error pat.	# of error bits	Sub-net's name	Mean square error	# of error pat.	# of error bits
1091	21.81	1091	26184	G_1	0.140833	6	6
				G_2	1.515929	41	49
				G_3	2.406807	55	76
				G_4	0.481865	10	10
				H	22.88248	1091	12437
2194	22.77	2194	52656	G_1	0.485220	16	16
				G_2	1.703482	44	55
				G_3	2.406832	51	73
				G_4	0.679743	16	18
				H	23.90936	2194	27199
4275	24.51	4275	102600	G_1	0.851415	100	119
				G_2	0.875400	50	67
				G_3	3.090423	232	407
				G_4	0.480404	6	6
				H	24.59527	4274	56820
8471	25.32	8471	203304	G_1	1.309214	135	170
				G_2	0.730064	52	69
				G_3	2.852241	221	379
				G_4	0.948055	45	52
				H	24.59993	8468	112910
16356	25.23	16356	392544	G_1	1.796017	861	1361
				G_2	1.310344	1006	1389
				G_3	1.376564	158	213
				G_4	1.167003	54	64
				H	25.390771	16318	239554

set. The results would have probably been improved if the number of neurons in hidden layers (and thus the number of interneuronal connections) had been increased.

Chapter 8

Conclusions

The objectives of the work were met successfully. The modularity has been studied in terms of entropy. The capabilities of neural networks and the process of training have been analyzed. The algorithms allowing modular design of neural networks have been suggested and successfully experimentally verified. A novel method of decomposition of neural networks helps to deal with systems of many inputs and outputs because such systems can be replaced with properly connected modules of sub-nets performing suitable partial calculations. Moreover, thanks to this approach, we can precisely identify the task that each sub-module is responsible for.

Decomposition of small and medium neural networks gave promising results. In the case of large problems – like air pollution data concerning a power plant – serial decomposition made neural network training possible and effective. The obtained results and related issues have been published in [53, 16, 42, 71, 72].

To sum up, modularity tends to be a promising way of designing neural networks and information systems, especially today, when they are present in almost every area of our life. Our cellular phones recognize our voice, the cameras and computers in security systems easily identify people entering the airport, computers defeat top chess players. Further development of technology and exponential growth of computational capabilities (Moor's law) of electronic devices will certainly allow the construction of currently unimaginable information systems, perhaps similar to real neural networks. Modularity will be unavoidable due to their complication. No one will think about separate neurons, weights, interconnections... I am deeply convinced that some levels of abstraction will be developed as it happened for example in the case of telecommunications, when the infrastructure became to complicated to be considered in details.

Certainly there is a need to develop a mature theory of information, perhaps not based on classical theories. I think this is the only way to design, if possible, a true artificial intelligence, and this is the field I would like to study in the future.

Bibliography

[1] A. Aho, J. Hopcroft, J. Ullman. Data Structures and Algorithms. Addison-Wesley,1983

[2] A. Aho, J. Hopcroft, J. Ullman. The Design and Analysis of Algorithms, Addison-Wesley, 1984

[3] http://www.altera.com

[4] K. Appel and W. Haken. Eyery planar map is four colorable. Part I. Discharging, *Illinois J. Math.* 21 (1977), pp. 429–490. Part II. Reducibility, *Illinois J. Math.* 21 (1977), pp. 491–567.

[5] R. Ahenhurst. The decomposition of switching functions. Int'1 Symp. on Theory of Switching Punct, pp. 74–116, 1959.

[6] R. Ash. *Real Analysis and Probability.* Accademic Press, New York, 1972.

[7] E. B. Baum, F. Wilczek. Supervised learning of probability distributions by neural. In D. A. Anderson *Neural information Processing Systems.* Accademic Institute of Physics, New York, 1988.

[8] T. C. Bartee. Computer design of multiple-output logical networks. *IRE Trans. on Elect. and Comp.*, pp. 21–30, March 1961.

[9] C. H. Bennett, D. P. DiVincenzo. Quantum information and computetion. *Nature*, 404:247–55, 2000.

[10] R. E. Blahut. *Principle and Practise of Information Theory.* Addison Wesley Publishing Company, 1987.

[11] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4), pp. 929–965, 1989.

[12] N. K. Bose, P. Linag. *Neural Network Fundamentals with Graphs, Algorithms, and Applications*. McGraw-Hill, Inc., 1996.

[13] R. Brayton, G. Hatchel, C. McMullen, A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, Boston, 1984.

[14] S. Brown, R. Francis, J. Rose, Z. Vranesic. *Field-Programmable Gate Arrays.* Kluwer Academic Publishers, 1992.

[15] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[16] P. Buciak, T. Łuba, H. Niewiadomski, M. Pleban, P. Sapiecha, H. Selvaraj. Decomposition and argument reduction of neural networks. IEEE Sixth International Conference on Neural Networks and Soft Computing (ICNNSC'02), Zakopane, Poland, June 11-15, 2002.

[17] P. Buciak, H. Niewiadomski, M. Pleban, H. Selvaraj, P. Sapiecha. The Variable Ordering for ROBDD/ROMDD-based Functional Decomposition. PDS 2001, IFAC Workshop Gliwice, Poland, 2001.

[18] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics and Operations Research* 4, pp. 233–235, 1979.

[19] H. Curtis. A generalized tree circuit. *Journal of the ACM*, 8, pp. 484–496, 1961.

[20] G. Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals & Systems*, 2, 4, pp. 303–314, 1989.

[21] G. Deco, D. Obradovic. *An information-theoretic approach to neural computing.* Springer-Verlag, Berlin, 1996.

[22] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Roy. Soc. (Lond.)*, A400, pp. 97–117, 1985

[23] A. Ehrenfeucht, D. Haussler. Learing decision trees from random examples. *Information and Computation* 82, pp. 231–246, 1989.

[24] R. P. Feynman. Simulating physics with quantum computers. *Int. J. Theor. Phys.*, 21(6/7), pp. 467–488, 1982.

[25] R. P. Feynman. Quantum mechanical computers. *Foundations of Physics*, 16(6), pp. 507–531, 1986.

[26] T. L. Fine. *Feedforward Neural Network Methodology.* Springer-Verlag, 1999.

[27] M. Fujita, Y. Matsunaga, K. Kakuda. On variable ordering of binary decision diagrams. *Proc. of European Design Automation Conference EDAC*, 1991.

[28] B. Girau. FPNA: Interaction between FPGA and neural computation. *International Journal of Neural Networks*, Vol. 10, No. 3, pp. 243–259, June 2000.

[29] S. Hawking. *A Brief History of Time. From the Big Bang to black Holes.* A Bantam Books, New York, 1988.

[30] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, 1990.

[31] A. Hyvärinen. Independent Component Analysis: Algorithms and Applications. *Neural Networks*, 13(4-5), pp. 411–430, 2000.

[32] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchange of variables. *Proc. International Conference on CAD*, pp. 472–475, 1991.

[33] I. T. Jollife. *Princple Component Analysis.* Springer-verlag, 1986.

[34] A. Karbowski, E. Niewiadomska-Szynkiewicz. *Obliczenia równolegle i rozproszone.* Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2001.

[35] M. Karpinski, A. Macintyre. Polynomial bounds for VC dimension of sigmoidal neural networks. *Proc. of the 27th Annual ACM Symposium on the Theory of Computing*, pp. 200–208, 1995.

[36] K. Kiviluoto, E. Oja. Independent component anaalysis for parallel time series. *Proc. ICONIP'98*, vol. 2, pp. 895–898, Tokyo, Japan, 1998.

[37] P. Koiran, E. D. Sontag. Neural Networks with Quadric VC Dimension. *Journal of Computer and System Sciences*, 54(1), pp. 190–198, 1997.

[38] S. Y. Kung. *Digital Neural Networks.* PTR Prentice-Hall, Inc., 1993.

[39] Y.-T. Lai, K.-R. R. Pan, M. Pedram. OBDD-Based Function Decomposition and its use in Digital Circuit Synthesis. *VLSI Design*, Vol. 3, nos. 3–4, pp. 225–248, 1995.

[40] T. Łuba. Decomposition of Multiple-Valued Functions. *Proc. IEEE-ISMVL*, USA, 1995.

[41] T. Łuba, H. Selvaraj, M. Nowicka, A. Krasniewski. Balanced Multilevel Decomposition and its Applications in FPGA-based Synthesis. In: G. Saucier, A. Mignotte (Eds.), *Logic and Architecture Synthesis*, Chapmann&Hall, 1995.

[42] T. Łuba, H. Niewiadomski, M. Pleban, H. Selvaraj, P. Sapiecha. Functional Decomposition and its Applications in Design of Digital Circuits and Machine Learning. IASTED International Conference, Applied Informatics, 2001.

[43] W. Maass. On the complexity of learning on feed forward neural networks. Institute for Theoretical Computer Science, Technische Universitaet Graz., 1993.

[44] W. Maass. Neural nets with superlinear VC-dimension. *Neural Computation*, 6(5), pp. 877–884, 1994.

[45] A. Macintyre, E. D. Sontag. Finitness results for sigmoidal neural networks. *Proc. of 25th Annual ACM Symposium on the Theory of Computing*, pp. 325–334, ACM Press, 1993.

[46] E. McCluskey. Minimization of Boolean Functions. *The Beli System Technical Journal*, Vol. 35, pp. 1417–1444, November 1959.

[47] C. McMullen, J. Shearer. Prime implicants, minimum covers, and the Complexity of logic simplification. *IEEE Transaction on Computing*, Vol. C-35, pp. 761–762, Aug. 1986.

[48] R. Moddemeijer, B. Spaanenburg. A decomposition technique for modular neural networks. *Proc. of the 11th workshop on Circuits, Systems and Signal Processing, IEEE/ProRISC*, pp. 427-433, ISBN: 90-73461-24-3, Utrecht, 2000.

[49] B. K. Natarajan. On learning sets and functions. *Machine Learning*, 4(1), 1989.

[50] B. K. Natarajan. *Machine Learning. A Theorethical Approach.* Morgan Kaufmann, San Mateo, California, 1991.

[51] P. S. Neelakanta. *Information-Theoretic Aspects of Neural Networks.* CRC Press, 1999.

[52] M. A. Nielsen, I. L. Chuang. *Quantum Computetion and Quantum Information.* Cambridge University Press, 2000.

[53] H. Niewiadomski, P. Buciak, M. Pleban, H. Selvaraj, P. Sapiecha, T. Łuba. Decomposition of Large Neural Networks. IASTED International Conference, Applied Informatics 2002, Austria, 2002.

[54] H. Niewiadomski, M. Pleban, P. Buciak, P. Sapiecha. Dekompozycja układów cyfrowych i sieci neuronowych. RUC, Poland, 2001.

[55] Z. Pawlak. Rough classification. *International Journal of Man-Machine Studies*, 20, pp. 469–483, 1984.

[56] R. Penrose. *Shadows of the Mind. A Search for the Missing Science of Consciousness.* Oxford University Press. 1994.

[57] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Łuba, S. Grygiel, M. Nowicka, R. Maki, Z. Wang, J. S. Zhang. Decomposition of Multiple-Valued Relations, *Proc. ISMVL* 1997.

[58] M. Pleban, P. Buciak, H. Niewiadomski, H. Selvaraj, P. Sapiecha, T. Łuba. Functional decomposition based on multi-valued decision diagrams. IASTED AIA Malaga, Spain, September 2002.

[59] M. Pleban, H. Niewiadomski, P. Buciak, P. Sapiecha, S. Yanushkevich, V. Shmerko. Argument Reduction Algorithms for Multi-Valued Relations. IASTED International Conference on Artificial Intelligence and Soft Computing, Banff, Canada, 2002.

[60] M. Pleban, H. Niewiadomski, P. Buciak, P. Sapiecha, M. Czenko, S. Yanushkevitch, V. Shmerko. Algorytmy redukcji argumentów dla wielowartościowych relacji. RUC 2002, Polska.

[61] M. Pleban, H. Niewiadomski. Dekompozycja funkcjonalna i jej zastosowania w syntezie logicznej i inżynierii wiedzy. XXV Krajowa Konferencja Elektroniki i Telekomunikacji Studentów i Młodych Pracowników Nauki. WAT, Warszawa, 2000.

[62] W. Quine. The Problem of Simplifying Truth Functions. *American Mathematical Monthly*, Vol. 59, pp. 521–531, 1952.

[63] T. Ristaniemi, J. Joutsensalo. On the performance of blind source separation in CDMA downlink. *Proc. Int Workshop on Independent Component Analysis and Signal Separation (ICA'99)*, pp. 437–441, Aussois, France, 1999.

[64] R. Rivest. *Lecture Notes in Machine Learning.* http://theory.lcs.mit.edu/ mona/lectures.html.

[65] E. Ronco, H. Gollee, P. Gawthrop. Modular Neural Network and Self-Decomposition. Technical Report CSC-96012. Faculty of Engineering, Glasgow, Scotland, 1997.

[66] J. Rueckl, K. Cave. Why are 'what' and 'where' processed by separate cortical visual systems? a computational investigation. *Journal of Cognitive Neuroscience*, vol. 1, pp.171–186, 1989.

[67] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams, *Proc. IEEE International Conference on Computer-Aided Design*, pp. 42–47, 1993.

[68] W. Rudin. *Real and Complex Analysis.* McGraw-Hill, New York, 1987.

[69] A. Sakurai. Tighter bounds of the VC-dimension of three-layer networks. *Prooc. of World Congress on Neural Networks*, pp. 540–543, 1993.

[70] P. Sapiecha, H. Selvaraj, M. Pleban. Decomposition of Boolean Relations and Functions in Logic Synthesis and Data Analysis. *Lecture Notes in Artificial Intelligence*, Vol. 2005, Springer Verlag, 2001.

[71] H. Selvaraj, H. Niewiadomski, M. Pleban. H. Sapiecha. Decomposition of Digital Circuits and Neural Networks. World Multiconference on Systemics, Cybernetics and Informatics, USA 2001.

[72] H. Selvaraj, H. Niewiadomski, P. Buciak, M. Pleban, P. Sapiecha, T. Łuba. Implementation of Large Neural Networks using Decomposition. The 2002 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences, Las Vegas, USA, 2002.

[73] C. E. Shannon. A symbolic analysis of relay and swicthing circuits. *Trans. AIEE*, 57:713–723, 1939.

[74] C. E. Shannon. A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423, 623–656, 1948.

[75] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.*, 26(5), pp. 1484–1509, 1997.

[76] http://www-ra.informatik.uni-tuebingen.de/SNNS/

[77] E. D. Sontag. *VC Dimension of Neural Networks. Neural Networks and Machine Learning (C.M. Bishop, ed.)*, pp. 69–95, Springer-Verlag, Berlin, 1998.

[78] E. D. Sontag. Feed forward nets for for interpolation and classification. *J. Comp. Syst. Sci.*, 45, pp. 20–48, 1992.

[79] W. Stallings. *Computer Organization and Architecture.* Prentice-Hall, Inc., a Simon & Schuster Company, 1996.

[80] W. Syslo, N. Deo, J. Kowalik. *Discrete Optimization Algorithms.* Prentice-Hall Endewood Cliffs, NJ, 1993.

[81] S. Tani, K. Hamaguchi, S. Yajima. The Complexity of the optimal variable ordering problems of shared binary decision diagrams. *Proc. International Symposium of Algorithms and Computation*, Vol. 762 of *Lecture Notes in Computer Science*, pp. 389–398, Springer, 1993.

[82] http://www.top500.org

[83] L. G. Valiant. Deductive learning. *Phil. Trans. Roy. Soc. Lond. A*, 312, pp. 441–446, 1984.

[84] V. N. Vapnik. *Estimation of Dependencies Based of Empirical Data.* Springer-Verlag, New York, 1982.

[85] V. N. Vapnik, A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2), pp. 264–280, 1971.

[86] R. Vigário, V. Jousmäki, M. Hämäläinen, R. Hari, E. Oja. Independent component analysis for identyfication of artifacts in magnetoencephalographic recordings. *Advances in Neural Information Processing Systems*, 10, pp. 229–235, MIT Press, 1998.

[87] J. Waldemark, M. Millberg, T. Lindblad, K. Waldemark, V. Becanowic. Implementation of pulse coupled neural network in FPGA. *International Journal of Neural Networks*, Vol. 10, No. 3, pp. 243–259, June 2000.

[88] W. Wan, M. A. Perkowski. A New Approach to the Decomposition of Incompletely Specified Multi-Output Functions Based on Graph Coloring and Local Transformations and its Applications to FPGA Mapping. *Proc. European Design Automation Conference*, Hamburg, 1992.

[89] R. L. Watrous. A comparison between squared error and relative entropy metrics using several optimization algorithms. *Complex Syst.*, 6, pp. 495–505, 1992.

[90] http://www.xilinx.com