Extracting Knowledge from Neural Networks in Image Processing

Berend Jan van der Zwaag[1], Kees Slump[1], and Lambert Spaanenburg[2]

[1] Dept. of Electrical Engineering, University of Twente,Enschede, the Netherlands
[2] Dept. of Information Technology, Lund University,Lund, Sweden

Summary. Despite their success-story, artificial neural networks have one major disadvantage compared to other techniques: the inability to explain comprehensively how a trained neural network reaches its output; neural networks are not only (incorrectly) seen as a "magic tool" but possibly even more as a mysterious "black box." Although much research has already been done to "open the box," there is a notable hiatus in known publications on analysis of neural networks. So far, mainly sensitivity analysis and rule extraction methods have been used to analyze neural networks. However, these can only be applied in a limited subset of the problem domains where neural network solutions are encountered. In this chapter we propose a wider applicable method which, for a given problem domain, involves identifying basic functions with which users in that domain are already familiar, and describing trained neural networks, or parts thereof, in terms of those basic functions. This will provide a comprehensible description of the neural network's function and, depending on the chosen base functions, it may also provide an insight into the neural network's inner "reasoning." To illustrate our method, the elements of a feedforward-backpropagation neural network, that has been trained to detect edges in images, are described in terms of differential operators of various orders and with various angles of operation. The results are then compared with image filters known from literature, which we analyzed in the same way.

Key words: Neural networks, rule extraction, edge detection, digital image processing, gradient filters.

1 Introduction

In the past 20 years artificial neural networks have gained renewed popularity as "universal problem solvers." Thousands of articles are published on the subject every year. However, one of the strongest weaknesses of neural networks is their inexplicability. This is an important aspect of the functionality of any technology, as users will be interested in "how it works," before trusting it completely. In particular, in safety critical systems (e.g., airlines, power stations, hospitals) it is imperative to know the behavior of an application under all possible input conditions. How can users trust a machine if they do not know how it works or what its behavior will be

under extreme circumstances? Neural networks learn from examples, and examples of extreme circumstances are rare by their very nature.

Since the early development of artificial neural networks, but during the past decade in particular, researchers have tried to analyze trained neural networks to provide insight into their behavior. For certain applications and in certain problem domains this has been successful. In particular in decision making systems and other systems that can easily be expressed in sets of rules, great advances have been made by the development of so-called rule extraction methods [4]. Neural network systems with relatively few inputs can sometimes be analyzed by means of a sensitivity analysis [9], which is a nonparametric statistical analysis technique.

However, most neural network systems are so high-dimensional that an extracted rule base would become too large to be easily interpreted, or so nonlinear that a sensitivity analysis would only be valid for a small part of the input space. For this reason, we propose domain-specific neural network analysis methods that utilize domain-specific base functions [20] that are easy to interpret by the user and that can even be used to optimize neural network systems. An analysis in terms of base functions may also make clear how to (re)construct a superior system using those base functions, thus using the neural network as a construction advisor.

2 Knowledge Extraction from Neural Networks

Despite their success-story, neural networks have one major disadvantage compared to other techniques: the inability to explain comprehensively how a trained neural network reaches its output; neural networks are not only (incorrectly) seen as a "magic tool" but possibly even more as a mysterious "black box" [14]. This is an important aspect of the functionality of any technology, as users will be interested in "how it works" before trusting it completely. In particular, in safety critical systems (e.g., airlines, power stations, hospitals) [6] [13] it is imperative to know the behavior of an application under all possible input conditions.

The merits of the analysis of neural networks include [1]:

- provision of a "user explanation" capability
- extension of neural network systems to "safety-critical" application domains
- software verification and debugging of neural network components in software systems
- improving the generalization of neural network solutions
- data exploration and the induction of scientific theories
- knowledge acquisition for symbolic AI systems

These merits clearly indicate how users can benefit from the analysis of neural networks. They will see improved functionality, and using the results of the analysis they can determine which type of neural network is best suited for solving their particular problems.

Compared to the amount of literature on theory and applications of neural networks, information on the analysis of neural networks is scarce and mostly limited

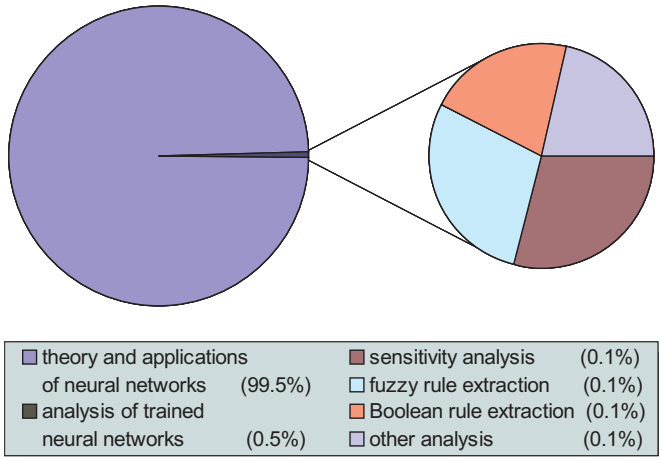


Fig. 1. Results of a search in several major patent databases, illustrating the relatively small amount of literature on the analysis of neural networks

to either *sensitivity analysis* or *rule extraction*. Sensitivity analysis [12] is a non-parametric statistical analysis technique, occasionally applied to neural networks, but more often appearing in other mathematical modeling and decision-making systems. Rule extraction [1] from neural networks originates from the field of symbol processing methods, which is rule-based rather than case-based. Neural network behavior described in sets of rules can provide insight into how the network comes to an answer. In Sections 2.1 and 2.2 these methods will be treated in more detail.

Literature also gives some alternative approaches. Feng [7] uses multi-layer perceptrons for parameter estimation and takes a first step to analyze the hidden units. Worth and Spencer [22] describe a neural network for tactile sensing. After learning, the weight vectors of all hidden units have the same structure, but the authors did not find any correlation between the relative sizes of the weight vectors and the outputs of the hidden units. VanderSteen [18] indicates that the correlation between weight changes provides more information. Mitchell [11] presents a method for interpreting the role of the hidden neurons geometrically. For the case in which function approximation is of most interest over a bounded region of the input space, the author shows that this interpretation may be used to check for redundancy among hidden units.

In general, neural networks with unsupervised training merely reorganize the input space by a winner-takes-all strategy, so analyzing them after training becomes fairly simple: an investigation into the reorganized input space reveals how the network has restructured the input space. Analyzing neural networks trained under supervision is far more complicated, as they operate by non-linear vector manipulation. They compromise rather than select and build a set of internal features that are not easily related to the input features.

2.1 Sensitivity Analysis

Sensitivity analysis (see, e.g., [12]) is a technique from the field of nonparametric statistical analysis. It is occasionally applied to neural networks, but more often in other mathematical modeling and decision-making systems. The general idea of sensitivity analysis is to investigate the effect that perturbations in the inputs have on the outputs, thus determining the sensitivity of the outputs to the inputs [10]. This is usually done for every input and every output, resulting in a matrix of sensitivities. These can then be used to determine whether any insignificant inputs can be ignored. In the neural network case, if a sensitivity analysis is performed for the hidden neurons, it could be used for pruning if some hidden neurons appear to have no influence on the network outputs [5]. Fu and Chen [8] use sensitivity analysis to investigate the generalization capability and error-correcting property of a neural network. Other examples of the use of sensitivity analysis for neural networks can be found in [3] and [9].

A weakness of sensitivity analysis applied to neural networks is formed by the fact that many applications of neural networks operate in high-dimensional input spaces, which would result in large sensitivity matrices that are hard to interpret. Another drawback is caused by the often strong nonlinearity of neural networks. This can cause outputs to have many different sensitivities over the full range of particular inputs, which in turn complicates the determination of the minimal value representation width or crisping [2]. Different analysis tools are needed in order to be able to analyze neural networks with high-dimensional input or output spaces or with strong nonlinear behavior. In fact, most neural networks have one or both of these properties, as they form some of the strengths of neural networks. The cases in which sensitivity analysis could be applied to neural networks are mainly those cases where the networks are small, those where sensitivity analysis is only applied to a (small) part of the neural network, and those where the dimensionality of the network has been greatly reduced. The latter case could, for example, be realized with modular neural networks such as in [15].

2.2 Rule Extraction

Rule extraction from neural networks (see, e.g., [1]) originates from the field of symbol processing methods, which is rule-based rather than case-based. Neural network behavior described in sets of rules can provide an insight into how the neural network comes to an answer. Craven and Shavlik [4] distinguish two approaches to testing rules for multilayer neural networks: the decompositional approach and the pedagogical approach (validity-interval analysis [17]). The decompositional approach is to extract rules for each hidden and output unit separately, thus providing a certain transparency, whereas the pedagogical approach enables the extraction of rules that directly map inputs to outputs for a network as a whole, thus basically not opening the “black box.” Pedagogical techniques are typically used in conjunction with a symbolic learning algorithm and the basic motif is to use the trained neural network to generate examples for the learning algorithm.

Andrews et al. [1] classify rule extraction techniques into the following categories:

- Boolean rule extraction using decompositional approaches;
- Boolean rule extraction using pedagogical approaches;
- Extraction of fuzzy rules.

Due to its nature, Boolean rule extraction is mainly used in problems with discrete-valued features. Fuzzy rule extraction can be applied to both problems with discrete-valued features and those with real-valued features. Crisping is required to discretisize the value space, but such may shudder the carefully constructed balance, that comprises the internal knowledge storage [2]. Furthermore, there are problem domains where solutions cannot easily or comprehensively be described in sets of rules or decision trees, due to, e.g., high dimensionality of the input space or very large sets of independent features. In those domains, different tools for analysis are needed, either in conjunction with rule extraction or separately.

2.3 Domain-Specific Base Functions

So far, mainly sensitivity analysis and rule extraction methods have been used to analyze neural networks. However, the previous sections make clear that these can only be applied in a limited subset of the problem domains where neural network solutions are encountered.

We need to investigate in which problem domains trained neural networks cannot satisfactorily be analyzed with sensitivity analysis or rule extraction techniques, and we need to develop new methods to analyze trained neural networks in those domains.

We therefore propose a method which, for a given problem domain, involves identifying basic functions with which users in that domain are already familiar, and describing trained neural networks, or parts thereof, in terms of those basic functions. This will provide a comprehensible description of the neural network's function and, depending on the chosen basic functions, it may also provide an insight into the network's inner "reasoning."

This concept is not so surprising, as many applications are based on a small number of arithmetic primitives. For instance, Ter Brugge et al. [16] describe the development of a stitch-weld tester. After a lengthy evaluation of various input features for the neural classifier, the acceptable solution is still reflecting the physical principle of operation: a dampened oscillation caused by reflections over different path lengths. In other words: it is based on a sine-wave and an exponential decay function.

Domain-specific analysis of neural networks through base functions not only provides insight into the in- and external behavior of neural networks and shows their possible limitations in particular applications, but it also lowers the acceptability threshold for future users unfamiliar with neural networks. Further, neural network analysis methods that utilize domain-specific base functions can be used to optimize neural network systems. An analysis in terms of base functions may even make clear how to (re)construct a superior system using those base functions, thus using the

neural network merely as a construction advisor. If a user does not want to trust a neural network for any reason whatsoever, he may still trust a non-neural system that would have been nearly impossible to construct without using a neural network as an advisor.

The idea of describing a trained neural network in terms of basic domain-specific functions was introduced and presented in earlier publications [20, 21]. For many problems in certain domains, such as linguistics and decision theory, the common, domain-dependent base functions could be chosen to be *if–then* rules or decision trees, in which case the analysis reduces to traditional rule extraction. Table 1 lists a few more problem domains where neural networks have been successfully applied. For each of these domains, possible base functions are presented. In the following we show for one such domain, i.e. digital image processing, how our analytical method can overcome the impracticality of more common knowledge extraction methods. In the case of edge detection – a digital image processing technique – the user will be familiar with image filters, particularly 2-dimensional differential operators. Hence, a description in terms of these digital image operators will enhance the understanding of the neural net's functionality. Therefore, we will illustrate the analysis of feed-forward–error-back-propagation neural networks trained for edge detection.

Table 1. Some application domains with potential domain-specific base functions

application domain	potential base functions	remarks
control theory	PID controllers	
decision theory	(fuzzy) if-then rules	i.e., classical rule extraction
signal processing (1-D)	basic operational filters	
digital image processing (2-D)	differential operators	
chemical process identification	molecular formulae	
general classification problems	feature map regions	cp. self-organizing map

3 Digital Image Processing

In this section we treat several basic techniques commonly used in digital image processing, or more specifically, in pixel classification. Edge detection (Section 3.1), line detection (Section 3.2), and spot detection (Section 3.3) are often a first step in "higher-level" techniques such as image classification, image segmentation, image enhancement, and others. Section 3.4 deals with the more generic class of differential operators, or gradient filters, which can also be used to detect edges, lines, and spots.

3.1 Edge Detection

Edge detection is frequently used in image segmentation. In that case an image is seen as a combination of segments in which image data are more or less homoge-

neous. Two main alternatives exist to determine these segments: (1) classification of all pixels that satisfy the criterion of homogeneousness; (2) detection of all pixels on the borders between different homogeneous areas. To the first category belong pixel classification (depending on the pixel value the pixel is part of a certain segment) and region growing methods. The second category is edge detection.

In fact, edge detection is also some sort of pixel classification: every pixel is either part of an edge or not. All edges together form the contours of the segments. After edge detection sometimes edge linking is used, in order to try to get the contours closed, as in practice not all pixels will be classified correctly, due to noise, etc. Many edge detection filters only detect edges in certain directions, therefore combinations of filters that detect edges in different directions are often used to obtain edge detectors that detect all edges.

In digital image processing, we can write an image as a set of pixels $f_{n,m}$ and an edge detection filter which detects edges with direction ϕ as a (template) matrix with elements $w_{k,l}$, see Fig. 2. We can then determine whether a pixel $f_{n,m}$ is an edge pixel or not, by looking at the pixel's neighborhood, see Fig. 3, where the neighborhood has the same size as the edge detector template, say $(2K+1)\times(2L+1)$. We then calculate the discrete convolution

$$g_{n,m} = \sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l} f_{n-k,m-l}, \tag{1}$$

where $f_{n,m}$ can be classified as an edge pixel if $g_{n,m}$ exceeds a certain threshold and is a local maximum in the direction perpendicular to ϕ in the image $g_{n,m}$.

Some examples of templates for edge detection are:

$$\underset{\text{Sobel, } 0^\circ}{\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}}, \quad \underset{\text{Kirsch, } 45^\circ}{\begin{bmatrix} -5 & -5 & 3 \\ -5 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}}, \quad \underset{\text{compass, } 90^\circ}{\begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix}}.$$

The dependency on the edge direction ϕ is not very strong; edges with a direction $\phi \pm 45^\circ$ will also activate the edge detector, though less strongly.

3.2 Line Detection

A similar filter is the line detector, which detects lines rather than edges. In fact, a line can be seen as two edges lying parallel and close to each other. Actually, most line detectors will also detect some edges, and most edge detectors will also detect some lines. In practice, there is not a well-defined boundary between lines and edges. For example, one could think of a line between two image segments (a line on an edge), or a narrow image segment with edges on either side (how narrow would it need to be to be regarded as a line rather than as a segment?).

Some simple line detector templates are:

$$\underset{\text{horizontal, } 0^\circ}{\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}}, \underset{\text{vertical, } 90^\circ}{\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}}, \underset{\text{diagonal, } 45^\circ}{\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}}, \underset{\text{diagonal, } -45^\circ}{\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}}.$$

Combining these four templates would result in a detector capable of detecting lines in practically all directions. Only strongly curved or crossing lines might not be detected equally well.

3.3 Spot Detection

A third filter type is the spot detector. A spot is an image segment approximately the size of a pixel. This of course depends on the image resolution; an object that shows as a spot in a low-resolution image would appear larger in a higher resolution, and an object that shows as a spot in a high-resolution image might very well be invisible in a lower-resolution image of the same scene.

Example templates for spot detection are:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 24 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 2 & 1 & -1 \\ -1 & 2 & 4 & 2 & -1 \\ -1 & 1 & 2 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}.$$

The templates on the right are slightly less sensitive to noise than the ones on the left, because of the difference in the neighborhood sizes. Nevertheless, these templates are quite sensitive to noise, which is easy to understand, given that noise often appears as spots. Because spots have no direction, the above templates are omnidirectional. This will be confirmed by the analysis results in Section 6.

Spot detectors can also be used to detect lines, because lines have similar properties as spots. Theoretically seen, spots have no width and no length, and lines have no width. However, whereas in Section 3.2 we needed a combination of several directional templates to detect lines in all directions, if would be sufficient to use only one omnidirectional spot detection template to detect lines in all directions. The reason why directional line detectors are often preferred lies in the larger noise sensitivity of the spot detection filters.

3.4 Differential Operators

A more generic type of image filters are the differential operators. Usage of these operators is based on the detection of changes in greylevel in a greylevel image. The gradient vector of a 2-dimensional continuous image $f(x, y)$ is defined as

$$\vec{\nabla} f(x,y) = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} & \dfrac{\partial f(x,y)}{\partial x} \end{bmatrix}^T = \begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix}. \tag{2}$$

For discrete images this can be seen as a template $[-1\ 1]$ for the gradient in the horizontal direction and as a template $\left[\begin{smallmatrix}1\\-1\end{smallmatrix}\right]$ for the gradient in the vertical direction. The gradients in the diagonal directions can be determined with Roberts' templates $\left[\begin{smallmatrix}-1 & 0\\ 0 & 1\end{smallmatrix}\right]$ and $\left[\begin{smallmatrix}0 & 1\\ -1 & 0\end{smallmatrix}\right]$ for gradients in -45° and $+45^\circ$ directions, respectively. Note that the direction of the edges detected by a first-order differential operator is perpendicular to the direction of the gradient. The same holds for the lines detected by a second-order differential operator.

4 Describing a Template in Terms of Differential Operators

We will now describe how an arbitrary image filter in matrix form can be seen as a composition of several differential operators, where the operators are of varying orders and operate in varying directions. In order to transform a template into a set of gradient filters, we first calculate the Taylor series expansion of the Fourier transformed template, and then we apply the inverse Fourier transform to get a description of the template which gives us knowledge about the differential components. The reason for using a Fourier transformation lies in the fact that a Fourier transformed filter description consists of a series of sinusoidals, which are easily differentiated to determine the Taylor components.

4.1 The Template in the Spatial Domain

In digital image processing, we can write an image filter as a (template) matrix with elements $w_{k,l}$, see Fig. 2.

$$\begin{bmatrix} w_{-K,L} & \cdots & w_{K,L} \\ \vdots & \ddots & \vdots \\ & w_{0,0} & \\ \vdots & \ddots & \vdots \\ w_{-K,-L} & \cdots & w_{K,-L} \end{bmatrix}$$

Fig. 2. A $(2K+1)\times(2L+1)$ template $w_{k,l}$

4.2 The Input Image in the Spatial Domain

We can write an image as a matrix of pixels $f_{n,m}$, see Fig. 3.

4.3 Filter Operation in the Spatial Domain

As already stated in Section 3.1, in pixel classification a filter operation can be defined by the discrete convolution of the filter template $[w_{k,l}]$ with the local neighborhood around a pixel $f_{n,m}$ in the input image $[f_{n,m}]$:

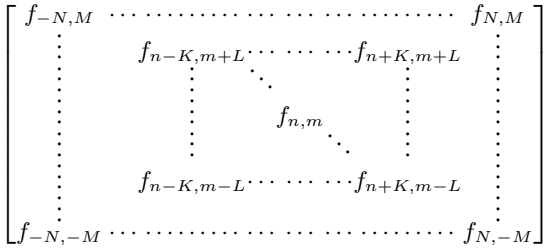


Fig. 3. A $(2N+1)\times(2M+1)$ image with a $(2K+1)\times(2L+1)$ neighborhood around $f_{n,m}$

$$g_{n,m} = \sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l} f_{n-k,m-l}, \tag{3}$$

where $g_{n,m}$ is the output pixel when the filter template $[w_{k,l}]$ is applied to the input image pixel $f_{n,m}$.

4.4 From Discrete to Continuous in the Spatial Domain

To facilitate Fourier transformation, two-dimensional Dirac functions (δ), defined as

$$\begin{gathered} \delta(x,y) = \begin{cases} 0 & \text{for } x^2+y^2 \neq 0 \\ \infty & \text{for } x^2+y^2 = 0 \end{cases}; \\ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(x,y)\,dx\,dy = 1, \end{gathered} \tag{4}$$

are assigned to the pixels $f_{n,m}$ and to the template elements $w_{n,m}$. This gives the continuous equivalent of the image $[f_{n,m}]$ as

$$f(x,y) = \sum_{n=-N}^{N} \sum_{m=-M}^{M} f_{n,m}\delta(x-n\Delta_x, y-m\Delta_y), \tag{5}$$

where δ is the Dirac function and Δ_x and Δ_y are the respective sampling periods in x- and y-direction. In the following, we will assume $\Delta_x=\Delta_y=\Delta$.

Similarly, we get the continuous equivalent of the filter template $[w_{n,m}]$:

$$w(x,y) = \sum_{n=-N}^{N} \sum_{m=-M}^{M} w_{n,m}\delta(x-n\Delta, y-m\Delta). \tag{6}$$

With the continuous equivalents of $[f_{n,m}]$ and $[w_{n,m}]$, the continuous equivalent of (3) becomes

$$g(x,y) = \int_t \int_s w(s,t) f(x-s, y-t)\,ds\,dt. \tag{7}$$

4.5 From Space to Frequency: Fourier Transformation

Two-dimensional Fourier transformation is defined as

$$H(u,v) = \mathcal{F}\{h(x,y)\} \equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y)e^{-2\pi j(xu+yv)}dx\,dy. \tag{8}$$

Using the above definition and the fact that

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y)\delta(x-a,y-b)dx\,dy = h(a,b), \tag{9}$$

the Fourier transformed image becomes

$$F(u,v) = \mathcal{F}\{f(x,y)\} = \sum_{n=-N}^{N}\sum_{m=-M}^{M} f_{n,m}e^{-2\pi j\Delta(nu+mv)}. \tag{10}$$

The transformed image is a continuous function in the frequency domain. The Fourier transformation $G(u,v)$ of the edge map gx,y is calculated similar to the Fourier transformed input image $F(u,v)$. If we write the filter function $W(u,v)$ as the transformation of the template $w_{n,m}$,

$$W(u,v) = \sum_{n=-N}^{N}\sum_{m=-M}^{M} w_{n,m}e^{-2\pi j\Delta(nu+mv)}, \tag{11}$$

then

$$G(u,v) = W(u,v)F(u,v). \tag{12}$$

This equation describes the filter operation in the frequency domain. A convolution in the space domain is transformed into a simple multiplication in the frequency domain.

According to (11), the filter function $W(u,v)$ in the frequency domain is a summation of weighted exponentials. This facilitates an easy Taylor series expansion.

4.6 Taylor Series Expansion

The Taylor series expansion of a function $h(u,v)$ around $(0,0)$ is defined as

$$h(u,v) = \sum_{r=0}^{\infty}\frac{1}{r!}\sum_{i=0}^{r}\binom{r}{i}u^i v^{r-i}\left.\frac{\partial^r h(u,v)}{\partial u^i \partial v^{r-i}}\right|_{(0,0)}. \tag{13}$$

Applying this definition to (11), the Taylor series of the filter function $W(u,v)$ in the frequency domain becomes

$$W(u,v) = \sum_n \sum_m w_{n,m} \sum_{r=0}^{\infty} \sum_{i=0}^{r} \frac{u^i v^{r-i}}{i!(r-i)!} (-2\pi j \Delta)^r n^i m^{r-i}, \tag{14}$$

with which the transformed edge map $G(u,v)$ can be written as

$$\begin{aligned} G(u,v) &= \sum_n \sum_m w_{n,m} \sum_{r=0}^{\infty} \sum_{i=0}^{r} \frac{u^i v^{r-i}}{i!(r-i)!} (-2\pi j \Delta)^r n^i m^{r-i} F(u,v) \\ &= \sum_{r=0}^{\infty} \sum_{i=0}^{r} (-2\pi j u \Delta)^i (-2\pi j v \Delta)^{r-i} \frac{F(u,v)}{i!(r-i)!} \sum_n \sum_m w_{n,m} n^i m^{r-i}. \end{aligned} \tag{15}$$

4.7 From Frequency Back to Space: Inverse Fourier Transformation

Equation (15) describes the output edge map, transformed to the frequency domain. If we now want a similar description of the edge map in the space domain, i.e., expansion into gradients of different orders, inverse Fourier transformation is required. This yields a Taylor series equivalent of $g(x,y)$:

$$g(x,y) = \sum_{r=0}^{\infty} \sum_{i=0}^{r} \frac{\partial^r f(x,y)}{\partial x^i \partial y^{r-i}} \frac{(-1)^r}{i!(r-i)!} \sum_n \sum_m w_{n,m} n^i m^{r-i}. \tag{16}$$

From (16) it can be deduced that the image filter described by $w_{n,m}$ can be regarded as composed of a series of differential operators, with the following continuous analogon:

$$g(x,y) = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \alpha_{i,j} \frac{\partial^{i+j}}{\partial x^i \partial y^j} f(x,y), \tag{17}$$

with $i+j=r$ and

$$\alpha_{i,j} = \frac{1}{i!j!} \sum_{n=-N}^{N} \sum_{m=-M}^{M} w_{n,m} n^i m^j \tag{18}$$

being the coefficients for the gradient vectors for the various values of i (order in x-direction) and j (order in y-direction).

4.8 Coordinate Transform

For a better insight in the types of differential operators, it can be determined if a filter is directional, and if so, what its main direction of operation is. To this purpose, the x- and y-axes can be rotated over an angle θ to new coordinate axes, see Fig. 4. Now x and y can be written as functions of ξ and η, depending on θ:

$$\begin{aligned} x &= x_\theta(\xi,\eta) = \xi \cos\theta - \eta \sin\theta \\ y &= y_\theta(\xi,\eta) = \xi \sin\theta + \eta \cos\theta \end{aligned} \tag{19}$$

With this transformation, $f(x,y)$ can also be written as

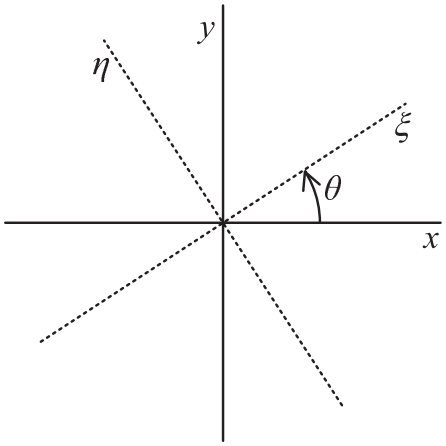


Fig. 4. Coordinate transformation from (x, y) to (ξ, η).

$$f(x,y) = f_\theta(x_\theta(\xi,\eta), y_\theta(\xi,\eta)) = f_\theta^{'}(\xi,\eta). \tag{20}$$

N.B. the notation $f_\theta^{'}$ has no relation to the gradient of f_θ.

We can now adapt (5)-(18) to the coordinate transformation of (19). The continuous image function $f_\theta^{'}(\xi,\eta)$ then becomes

$$f_\theta^{'}(\xi,\eta) = \sum_n \sum_m f_{n,m}\delta(\xi - n\Delta\cos\theta - m\Delta\sin\theta, \eta + n\Delta\sin\theta - m\Delta\cos\theta), \tag{21}$$

and its Fourier transformed version is then

$$F_\theta^{'}(\mu,\nu) = \sum_n \sum_m f_{n,m} \mathrm{e}^{-2\pi \mathrm{i}\Delta((n\cos\theta + m\sin\theta)\mu + (-n\sin\theta + m\cos\theta)\nu)}. \tag{22}$$

In a similar manner as in Section 4.5, the output map can be calculated as

$$G_\theta^{'}(\mu,\nu) = W_\theta^{'}(\mu,\nu)F_\theta^{'}(\mu,\nu), \tag{23}$$

with

$$W_\theta^{'}(\mu,\nu) = \sum_n \sum_m w_{n,m} \mathrm{e}^{-2\pi \mathrm{i}\Delta((n\cos\theta + m\sin\theta)\mu + (-n\sin\theta + m\cos\theta)\nu)} \tag{24}$$

being the Fourier transformed filter function in the transformed coordinate system.

We can now write the Taylor series expansion of the filter function $W_\theta^{'}(\mu,\nu)$ by applying the definition in (13) to (24):

$$\begin{aligned} W_\theta^{'}(\mu,\nu) = \sum_n \sum_m w_{n,m} \sum_{r=0}^{\infty} \sum_{i=0}^{r} \frac{\mu^i \nu^{r-i}}{i!(r-i)!} (-2\pi j\Delta)^r \\ \cdot (n\cos\theta + m\sin\theta)^i (-n\sin\theta + m\cos\theta)^{r-i} \end{aligned} \tag{25}$$

or, using the polynomial equality

$$(a+b)^i = \sum_{k=0}^{i} \binom{i}{k} a^k b^{i-k}, \tag{26}$$

we get

$$\begin{aligned} W'_\theta(\mu,\nu) = \sum_{r=0}^{\infty}\sum_{i=0}^{r}(2\pi j\mu\Delta)^i(2\pi j\nu\Delta)^{r-i}\frac{(-1)^{r-i}}{i!(r-i)!}\sum_n\sum_m w_{n,m} \\ \sum_{k=0}^{i}\binom{i}{k}(n\cos\theta)^k(m\sin\theta)^{i-k}\sum_{l=0}^{r-i}\binom{r-i}{l}(-n\sin\theta)^l(m\cos\theta)^{r-i-l}. \end{aligned} \tag{27}$$

Equation (27) is again a Taylor series description of the filter function in the frequency domain. Now the complete output edge map can again be calculated in the frequency domain and inverse Fourier transformation results in the following equation in ξ and η:

$$\begin{aligned} g'_\theta(\xi,\eta) = \sum_{r=0}^{\infty}\sum_{i=0}^{r}(-1)^r\frac{\partial^r f'_\theta(\xi,\eta)}{\partial\xi^i\partial\eta^{r-i}}\sum_n\sum_m w_{n,m} \\ \sum_{k=0}^{i}\sum_{l=0}^{r-i}\frac{(-1)^l n^{k+l}m^{r-k-l}}{k!l!(i-k)!(r-i-l)!}(\sin\theta)^{i-k+l}(\cos\theta)^{k+r-i-l} \end{aligned} \tag{28}$$

or

$$g'_\theta(\xi,\eta) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\beta_{\theta,i,j}\frac{\partial^{i+j}}{\partial\xi^i\partial\eta^j}f'_\theta(\xi,\eta), \tag{29}$$

with $i+j=r$ and

$$\begin{aligned} \beta_{\theta,i,j} = (-1)^{i+j}\sum_{n=-N}^{N}\sum_{m=-M}^{M} w_{n,m} \\ \sum_{k=0}^{i}\sum_{l=0}^{j}\frac{(-1)^l n^{k+l}m^{i+j-k-l}}{k!l!(i-k)!(j-l)!}(\sin\theta)^{i-k+l}(\cos\theta)^{j+k-l}. \end{aligned} \tag{30}$$

Equation (30) gives the Taylor series coefficients of the edge detection filter template, from which we can deduce of which orders of differential operators the filter consists, i.e., those i and j that give the larger $\beta_{\theta,i,j}$, and in which direction(s) these operators work optimally, i.e., the angle(s) θ for which $\beta_{\theta,i,j}$ is maximal for certain i and j. This can be represented graphically by drawing the value of $\beta_{\theta,i,j}$ as a function of θ for various i and j. See Figs. 9 and 12 for some examples. In these graphs, the normalized absolute value of $\beta_{\theta,i,j}$ as a function of θ is represented by the distance from the center of the graph in the direction of θ; positive values of $\beta_{\theta,i,j}$ are shown as thick dark lines, negative values as thin lines.

5 Neural Network Edge Detector

In order to test our analysis method, we trained several artificial neural networks for edge detection. The neural networks were of the feed-forward error-backpropagation type, with 3×3 to 11×11 inputs, 4 to 8 units in the single hidden layer, and a single output. All units used sigmoid activation functions. Some networks were trained with a training image containing sharp edges only (see Fig. 5). A different image was used as a test set, see Fig. 6 for a test result by a neural network edge detector. Other networks were trained with a similar image as the one in Fig. 5, but containing sharp edges as well as blurred ones, and that had Gaussian noise added to one half of the image. This made the neural network edge detectors less sensitive to noise, as also becomes visible in the analysis results shown hereafter in Section 6.

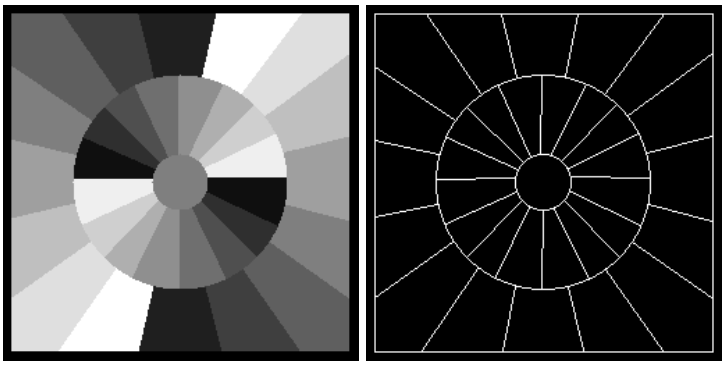


Fig. 5. A 128×128-pixel training image (*left*) and corresponding reference edge map (*right*)

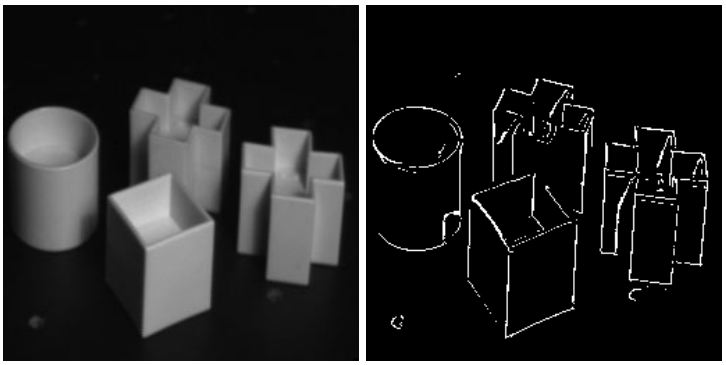


Fig. 6. Test image (*left*) and thresholded result after edge detection by a neural network (*right*)

6 Results

As (30) gives the Taylor series coefficients of any image filter template, we can apply it to existing edge detector templates as well as to a neural network edge detector's hidden units, whose weight matrices can also be regarded as templates.

First, we give brief analysis results for some of the line and spot detector templates shown in Sections 3.2 and 3.3. In the case of line-detecting templates, it is clear from the first template shown in Section 3.2, that it detects horizontal lines, and from the second one that it detects vertical lines, therefore it is not surprising that the second-order coefficients are strongest in the vertical and horizontal directions respectively, as seen in Fig. 7. The zero-order and first-order coefficients are all zero. This is also the case for the spot detector, see Fig. 8, except that in this case the second-order behavior is indeed omnidirectional, or rotation-invariant, as already indicated in Section 3.3.

Next, we show analysis results for the diagonal Kirsch template shown in Section 3.1. As can clearly be seen from Fig. 9, this template shows no low-pass (averaging) behavior, has a strong first-order gradient operation in diagonal direction, as could be expected, and weak second-order gradient behavior. A similar result is seen in Fig. 10, where a horizontal Kirsch template is compared with two other horizontal edge detection templates. The differences between these three are only visible in their second- (and higher-) order behavior.

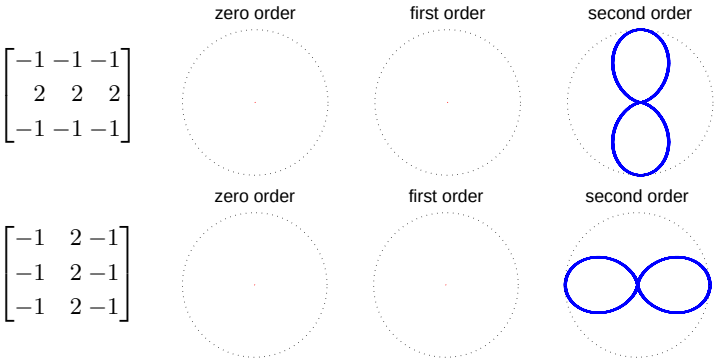


Fig. 7. Templates and low-pass, gradient, and second-order gradient analysis results for a horizontal (*top*) and a vertical (*bottom*) line detector

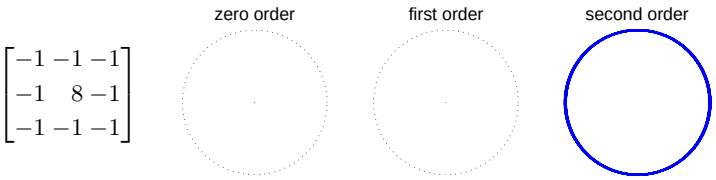
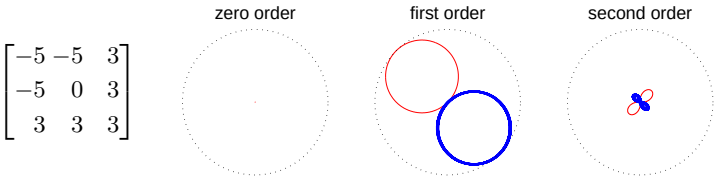
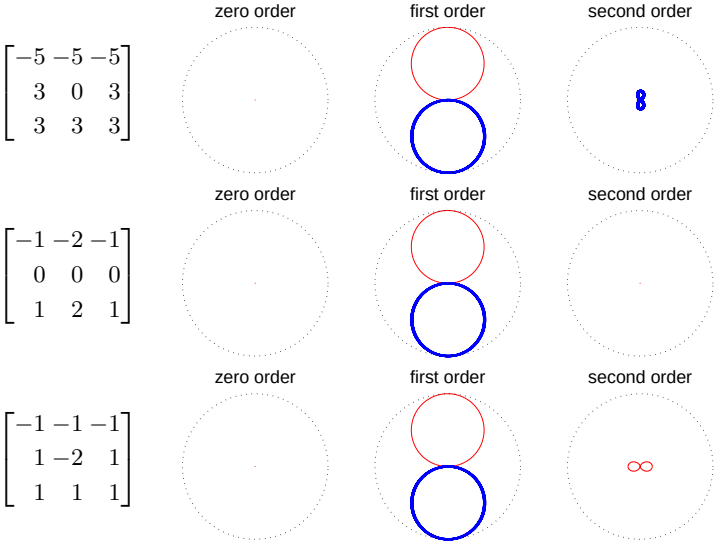


Fig. 8. Template and low-pass, gradient, and second-order gradient analysis results for a spot detector



$$\begin{bmatrix} -5 & -5 & 3 \\ -5 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

Fig. 9. Template and low-pass, gradient, and second-order gradient analysis results for a diagonal Kirsch edge detector template



$$\begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Fig. 10. Templates and low-pass, gradient, and second-order gradient analysis results for three horizontal edge detector templates: Kirsch (*top*), Sobel (*middle*), Prewitt compass (*bottom*)

Figure 11 shows results for the 4 hidden units of a small neural network edge detector, which was trained with the sharp edges shown in Fig. 5. For the purpose of showing the hidden units' weight values graphically, they have been scaled to values between -1 (black) and $+1$ (white). The weight templates shown in the first column of Fig. 11 already give some insight into their behavior, but the Taylor series coefficient analysis clearly shows that three of the units detect edges in various directions, and one unit acts as a second-order gradient filter with minor first-order gradient behavior. Notice that all four units do not have a significant low-pass (zero-order gradient) component.

Another neural network with the same architecture was trained with sharp, blurred, and noisy variants of the images shown in Fig. 5. The weight templates of the hidden units are shown in Fig. 12 along with the graphical representations of their Taylor series coefficients. This network's units have similar gradient components as the previous one, although the second-order gradient components are somewhat stronger. The low-pass components are significantly present, as compared to

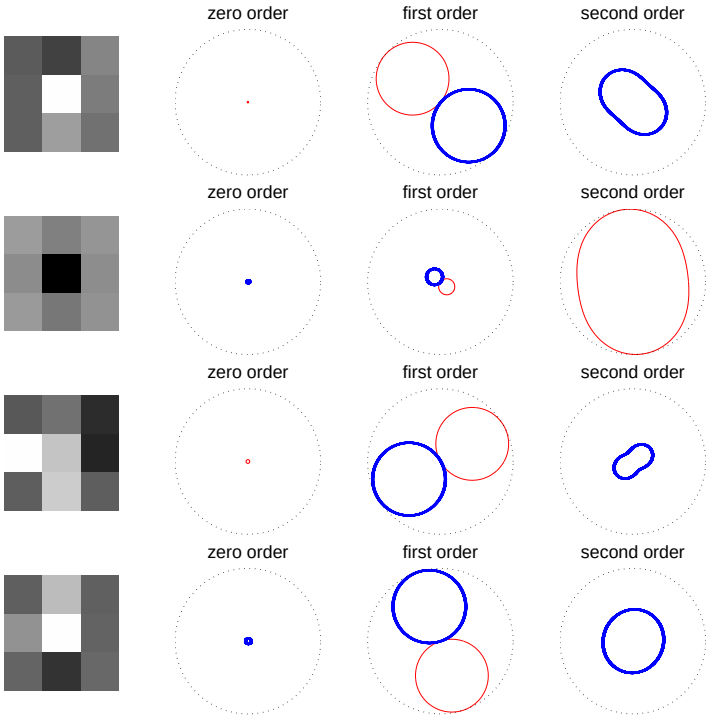


Fig. 11. Weight templates and low-pass, gradient, and second-order gradient analysis results for all 4 hidden units of a neural network edge detector with (3×3) inputs, 4 hidden units, and one output unit, trained with sharp edges

the previous network. This is a result of the different training. Low-pass or averaging behavior makes the network less sensitive to noise and improves the edge detection ability of the second network.

Although the above only gives some analysis results for the units in the hidden layer, it should be clear that a description of the neural network as a whole can be derived from these results. The weight between a hidden units and the output unit represents the "importance" of the hidden unit's edge detection outcome, which is then combined with the other hidden units' outcomes into a single answer indicating whether the pixel under investigation belongs to an edge or not.

Some larger neural networks have also been trained and analyzed, with similar results, although in general, the larger the network, the more variety in behavior among the neural units. In a few cases, certain units showed very strong higher-order behavior, indicating that those units functioned as noise detectors only. Although such units usually have weak connections to the output unit (low importance), removing them from the network (pruning) often results in worse edge detection capabilities for the network as a whole. This is because the noise detecting unit decreases the confidence of an edge detection outcome if the local neighborhood around the pixel under in-

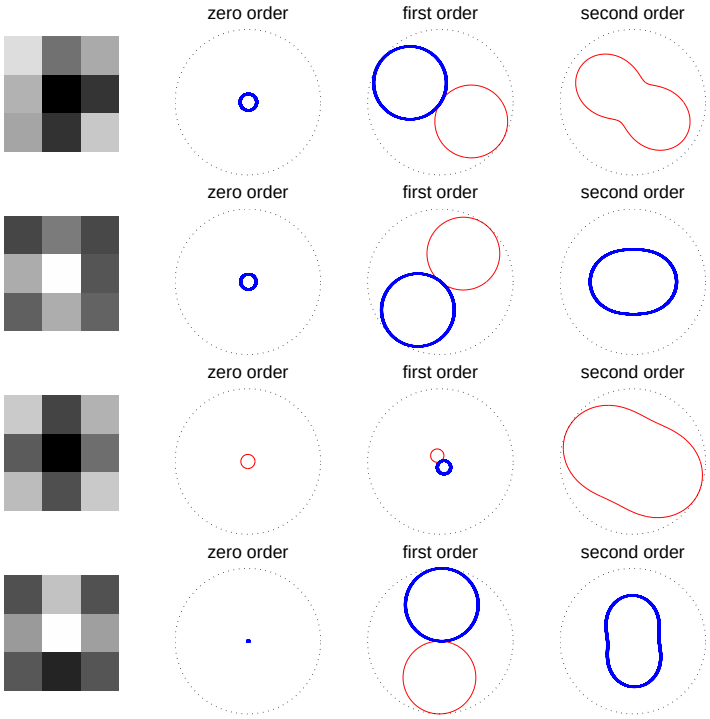


Fig. 12. Weight templates and low-pass, gradient, and second-order gradient analysis results for all 4 hidden units of a neural network edge detector with (3×3) inputs, 4 hidden units, and one output unit, trained with sharp/blurry/noisy edges

vestigation is very noisy. Units detecting sharp edges could easily misclassify such pixels as edge pixels.

7 Conclusions

We have trained neural networks to detect edges in digital images and analyzed them into gradient filter components. From the results displayed and described in the previous sections it is clear that it is indeed feasible to describe the trained neural networks in terms of basic functions from the image processing domain.

The description with gradient filter components gives easy insight into the behavior of the neural network as an edge detector, and allows simple comparison with other edge detectors which can in the same way be described in terms of gradient filter components.

In general, the analysis consists in describing the internal functionality of the neural network in terms of basic domain functions, functions that can be considered basic in the application domain of the neural network. This means that users who may not be familiar with artificial neural networks, but who are familiar with basic

functions that are often used in their problem domain, can gain insight in the way the neural network solves their problem. For such users, this is often an important factor in deciding to apply artificial neural networks to a problem that may be difficult to solve otherwise.

References

1. Andrews R, Diederich J, Tickle AB (1995) A survey and critique of techniques for extracting rules from trained artificial neural networks. Neurocomputing Research Centre report, Queensland University of Technology, Australia
2. Blattner J, Neuer S, Spaanenburg L, Nijhuis JAG (1993) Optimizing fuzzy rules by neural learning. In: Digest International Conference on Mathematical and Intelligent Models in System Simulation MISS'93 (Brussels) 238–246
3. Choi JY, Choi C-H (1992) Sensitivity analysis of multilayer perceptron with differential activation functions. IEEE Transactions on Neural Networks 3(1):101–107
4. Craven MW, Shavlik JW (1994) Using sampling and queries to extract rules from trained neural networks. In: Machine Learning: Proceedings of the Eleventh International Conference, San Francisco CA, USA
5. Engelbrecht AP, Cloete I (1996) A sensitivity analysis algorithm for pruning feedforward neural networks. In: IEEE International Conference on Neural Networks, (Washington) 2:1274–1277
6. ERA Technology Ltd. (1997) Neural networks producing dependable systems. ERA Report 97-0365, Leatherhead. Http://www.era.co.uk/techserv/pubs/p970365.htm (accessed 2 May 2001)
7. Feng T-J (1991) Backpropagation networks for parameter estimation (an overall report). Report, University of Twente, the Netherlands
8. Fu L, Chen T (1993) Sensitivity analysis for input vector in multilayer feedforward neural networks. In: Proceedings IEEE International Conference on Neural Networks (San Francisco CA) I:215–218
9. Hashem S (1992) Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions. In: Proceedings of the 1992 International Joint Conference on Neural Networks, IEEE Press, Piscataway, NJ, USA, 1:419–424
10. Klimasauskas CC (1991) Neural nets tell why. Dr. Dobbs's Journal, pp 16–24
11. Mitchell J (1992) A geometric interpretation of hidden layer units in feedforward neural networks. Network 3:19–25
12. Rios Insua D (1990) Sensitivity analysis in multi-objective decision making. Lecture Notes in Economics and Mathematical Systems 347. Springer Verlag, Berlin
13. Sharkey AJC, Sharkey NE, Gopinath OC (1995) Diversity, neural nets and safety critical applications. In: Niklasson LF, Boden MB (eds) Current trends in connectionism. Lawrence Erlbaum Associates, Hillsdale, NJ, pp 165–178
14. Sjoberg J, Zhang Q, Ljung L, Benveniste A, Deylon B, Glorennec P-Y, Hjalmarsson H, Juditsky A (1995) Nonlinear black-box modeling in system identification: a unified overview. Automatica 31(12):1691–1724
15. Spaanenburg L, Slump C, Venema R, van der Zwaag BJ (2002) Preparing for knowledge extraction in modular neural networks. In: Proceedings 3rd IEEE Benelux Signal Processing Symposium (SPS-2002), Leuven, Belgium

16. terBrugge MH, Nijhuis JAG, Jansen WJ, Drenth H, Spaanenburg L (1995), “On the representation of data for optimal learning. In: Proceedings ICNN’95 (Perth, Western Australia) VI:3180–3184
17. Thrun SB (1993) Extracting provably correct rules from artificial neural networks. Tech Report IAI-TR-93-5, Institut für Informatik III, Universität Bonn, Germany
18. van der Steen EW, Nijhuis JAG, Spaanenburg L, van Veelen M (2001) Sampling casts plasticity in adaptive batch control. In: Proceedings ProRISC’01 (Veldhoven, the Netherlands) 646–651
19. van der Zwaag BJ (1992) Analysis of neural network edge detectors. Report no. 92M185, University of Twente, the Netherlands
20. van der Zwaag BJ, Spaanenburg L, Slump C (2002) Analysis of neural networks in terms of domain functions. In: Proceedings IEEE Benelux Signal Processing Symposium SPS-2002 (Leuven, Belgium) 237–240
21. van der Zwaag BJ, Slump C, Spaanenburg L (2002) Process identification through modular neural networks and rule extraction. In: Ruan D, D’hondt P, Kerre EE (eds) Computational Intelligent Systems for Applied Research: Proceedings of the 5th International FLINS Conference (Ghent, Belgium), World Scientific, Singapore, pp 268–277
22. Worth AJ, Spencer RR (1989) A neural network for tactile sensing: the Hertzian contact problem. In: International Joint Conference on Neural Networks I:267–274