Archaeology and the application of artificial intelligence : case-studies on use-wear analysis of prehistoric flint tools

Dries, M.H. van den

6 A neural network prototype for use-wear analysis: WARP[1]

6.1 Introduction

In chapter 2 the neural network was introduced as a knowledge-based approach that is even more unknown and less applied by archaeologists than expert systems. This approach may, however, have at least an equal potential for our discipline. Neural networks are able to handle large quantities of complex data, of which the exact importance of the involved variables is unknown. With rapidly increasing success they are applied for a large variety of industrial, medical and other practical tasks (*e.g.* Kappen & Gielen 1997), like classifications, diagnosis, pattern recognition and prediction. Likewise, they may provide an appropriate solution to archaeological problems.

An artificial neural network is a computer program that has been developed to simulate the principle of the biological neural network in the human brain. By means of a mathematical model it tries to predict the outcome of highly complex situations by generalizing from similar situations that occurred previously. Such situations must be presented to a network in large quantities, because it must learn from these examples in order to be able to generalize from them. The learning process takes place during a training procedure in which it autonomously searches for the (non-linear) relations between the variables of the examples. It selects the variables that are most important and learns to ignore the ones which are irrelevant. The purpose of this is to model the relations between the input and output data, not to memorize the examples.

Apart from their background and aim, neural networks and expert systems have not much in common. They have a different architecture, they use specific knowledge storing and processing methods and are applicable to different kinds of tasks. Nevertheless there is some rivalry between the advocates of neural networks and of expert systems. Both present their approach as the most successful in simulating human reasoning.

In some instances, this rivalry even had its effect in host disciplines. For instance, the archaeologist Gibson published a paper in which he advocated the superiority of neural networks above expert systems regarding their functional abilities and social acceptability (Gibson 1992: 265) (see also chapter 2). The present case study was started as a reaction to this statement. The prime aim of this case study was to investigate the possibilities and difficulties of formalizing and modelling the expert knowledge that is involved in wear trace analysis by means of a neural network approach. The findings would be compared with my experiences regarding the development of WAVES in order to find arguments that would either support or reject Gibson's statement. The preliminary version of WAVES could serve a functional comparison with a neural network approach.

It was thought that use-wear analysis would be a suitable application area for a neural network application because it involves a task that this approach is good at: classification. The output of the network would give an indication of the class of patterns (the contact material) to which the input (wear attributes) probably belongs. Nevertheless, the aim of this study was not to develop a fully operational system which, like WAVES, would be useful for student training. It was merely meant as an initial exploration of this new technique. Consequently, the resulting neural network application (WARP) is only a limited prototype which has not (yet) been optimally designed and trained.

In this chapter, an introduction is given of the fundamentals of neural networks (paragraph 6.2) and of their development (6.3) first. Next, the composition of WARP will be discussed (paragraph 6.4), in particular its design and the way the knowledge was represented. Finally (paragraph 6.5), the functionality of WARP will be evaluated. It will be examined whether and in what way it can meet the requirements that archaeologists in general and use-wear analysts in particular pose on knowledge-based applications (see chapter 2).

Since neural networks may play an increasingly important role in future archaeological research, it has been tried throughout this chapter to illustrate the difference between this approach and expert systems. The aim is not to prove the superiority of one of the two, but rather to give an impression of their capabilities and of the purposes they can be employed for. For an objective assessment of the actual performance of WARP, the reader is referred to chapter 7, in which the result of a test is given that involved both a first version of WAVES and the neural network prototype.

6.2 Neural network fundamentals

6.2.1 HISTORICAL BACKGROUNDS

It is often thought that neural networks are one of the more recent developments in the field of artificial intelligence. This is a misunderstanding. In fact, neural networks originate from biological research on brain function rather than from artificial intelligence research and they experience a renewed interest. Their development already started more than 50 years ago when McCulloch and Pitts (1943) presented a model for *neuron functioning* that was thought to be representative of processes in the human brain. Neurons are the cells that the human brain consists of. Subsequently Hebb (1949) related neuron activation with learning processes and neuron distribution with knowledge storage. His rules are known as *Hebbian learning* and many of the present artificial neural networks are still being built according to his ideas.

It was only from 1956 onwards that computers were deployed to simulate human neural networks by means of artificial counter parts. The first applications were presented at the famous Dartmouth summer conference in that year. They marked the official starting date of the artificial intelligence research (*e.g*. Patterson 1996). During the 1960's the research on this subject blossomed. Especially Rosenblatt's so-called *perceptron networks* received much interest (Rosenblatt 1961). This lasted until Minsky and Papert (1969) mathematically analyzed the computational abilities of these networks and clearly expounded their restrictions. Simultaneously, their achievements became overshadowed by approaches that were based on rules instead of automated learning, such as expert systems. The latter were expected to offer more promising results in simulating human intelligence because their knowledge handling resembled heuristic reasoning. As a result, most research on neural networks was almost immediately abandonned.

It did not take long, however, before the limitations of the rule-based approaches were acknowledged as well. In the early 1980's, after less than 10 years of research, the expert system technology was no longer believed to be able to reveal the secrets of human reasoning (*cf*. Copeland 1993: 32). Gradually the conviction gained ground that the overwhelming data processing capabilities of the human brain had to be caused by massive *parallelism* rather than by sequential methods. Since parallelism was the very principle on which the early neural networks were based, several researchers returned to this approach and revived the research.[2]

The renewed research not only led to the development of new network models and knowledge processing algorithms that could meet some of the objections of the original neural networks, but the improvements and new achievements also started to direct the attention to commercial applications. Like expert systems, neural networks gradually became successfully employed in all kinds of domains. They turned out to perform well on tasks like pattern recognition (speech and image processing), trend or behavior prediction, noisy data filtering, non-linear data analysis, classification, diagnosis and control optimization (*cf*. Lawrence 1991: 13; Patterson 1996: 215). A well-known example of a subject that a neural network can handle successfully, is the prediction of price fluctuations on the stock market. This market is very vulnerable to changes in all kinds of variables, but it is hardly known which variables are influential to what degree and at what time.

Based on examples of previous situations and the resulting changes in the stock prices, a neural network is able to predict the results when similar or slightly deviating situations occur. These are tasks that are very hard to model and to represent by means of decision rules, but they can be handled perfectly by a neural network approach.

The revived interest in neural networks caused a competition between advocates of both approaches. Especially the devotees of the new neural network approach claimed to have found the most successful method for simulating human reasoning. They believed that the days of the sequential approaches had been counted (*e.g*. Copeland 1993: 242-245). However, considering the many dissimilarities and the fact that they serve different types of tasks, this competition is rather peculiar, and not very useful. In fact, there are so many differences between expert systems and neural networks that they can hardly compete with each other. For instance, neural networks suit problems that involve complex non-linear knowledge which cannot be captured by rules, while expert systems can only be applied for tasks that comprise of explicit deduction procedures. Since the former are able to autonomously detect hidden relations between the variables in a data set, they do not require intensive knowledge elicitation and modelling procedures (fig. 39). In other words, they only need examples which contain knowledge implicitly rather than explicit decision rules.

However, neural networks have not been developed to perform logical deductions or step-wise actions. Their processing characteristics are based on statistical calculations and the resulting predictions reflect estimations rather than mathematical facts (*cf*. Lawrence 1991: 18). Furthermore, the two methods work with different data formats. Expert systems can process non-numerical (symbolic) knowledge, while neural networks are based on numerical data. This implies that the latter can process information much faster. Additionally, networks owe their speed to the strategy of parallel processing.

Neural networks are also known to be more robust than traditional, sequentially oriented programs like expert systems. Within a neural network the knowledge is distributed

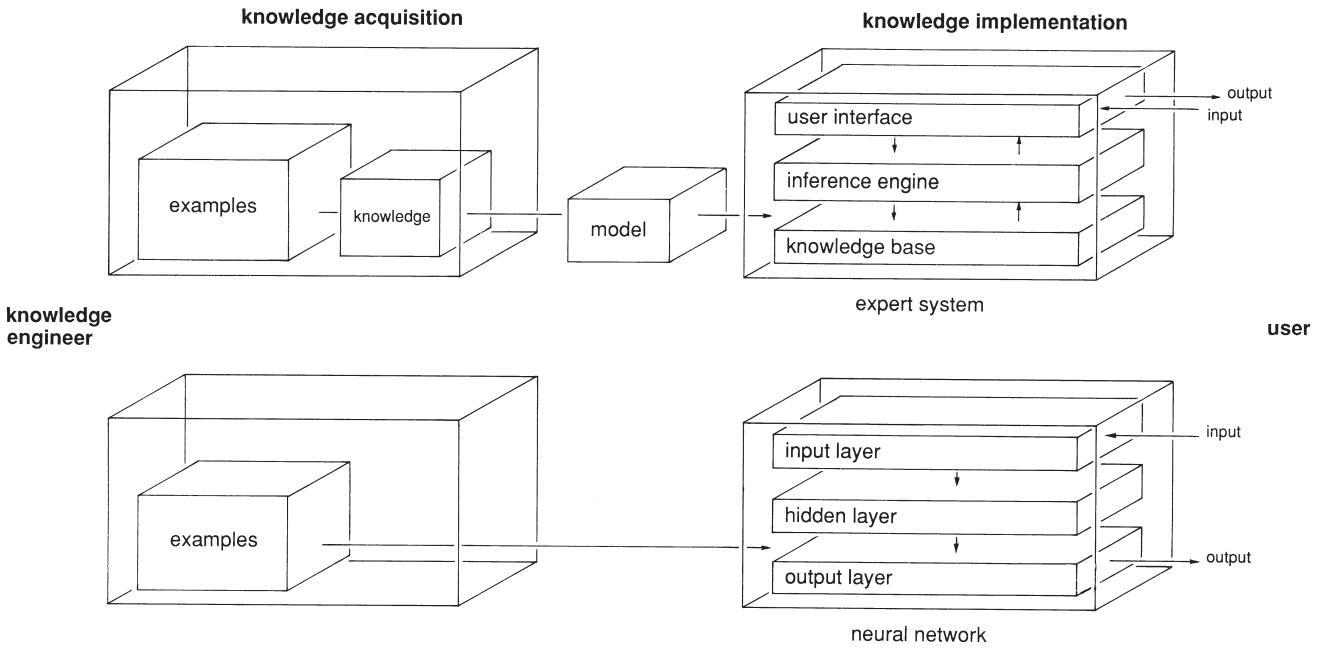

Fig. 39. A comparison of the structure (left side) of an expert system (top) with a neural network (below) and of the knowledge composition (right side). A neural network is only supplied directly with examples, whereas an expert system requires modelled and formalized knowledge.

across all of its neurons and connections, while the knowledge in an expert system is located in single rules. Moreover, the knowledge in a network, the examples, contains redundant information because it has not been disposed of irrelevant data. Consequently, a neural network is still able to perform when part of the network is damaged or when the training examples contain a considerable amount of noise. With expert systems this is impossible.
Those differences imply that the two techniques are meant to serve different purposes: expert systems for simulating heuristic methods and techniques, neural networks for detecting or recognizing (hidden) relationships between the properties that describe patterns within large and complex data-sets. When applied in this way, neural networks are able to perform significantly better than standard statistical techniques (Epping *et al*. 1993).

6.2.2 ARCHITECTURE[3]

Due to the fact that the research on neural networks aims to discover the secrets of the power of the human brain, the architecture of an artificial neural network imitates that of its biological example. The brain is a very complex organ that consists of ten to one-hundred billion cells, called *neurons*. These neurons are cells that are able to receive, store and send information. They are connected with each other through *dendrites*, hair-like channels through which the neurons can transmit signals. A single neuron may be connected to tens of thousands of other neurons and together they form a complex network. Via the dendrites the neurons send electrical and chemical signals in order to communicate with each other. Since these signals can be transferred simultaneously, thousands of impulses can be transported through the neurons per second. Therefore, the network structure in our brain enables a massive neuron activity. This implies that it can quickly process an enormous quantity of information. Without this, we would not be able to respond adequately to the signals from our sense organs and we would be very vulnerable in perilous situations.
Like the human brain, an artificial network consists of neurons, *i.e*., small processing elements that can receive, process and send signals. They can pass information to each other through programmed instructions. The network translates these relationships as connections between the neurons. Like in the human brain, the neurons in a neural network are arranged in layers and connected with each other, although not physically. Most neural networks consist of at least three layers: an *input layer*, one (or several) *hidden layer*(s), and an *output layer*. Each neuron in one layer is connected with those of another layer (fig. 40). In a neural network application, the neurons of the input layer represent the variables that describe a problem; the output neurons represent the solution that is associated with it. The neurons of the hidden

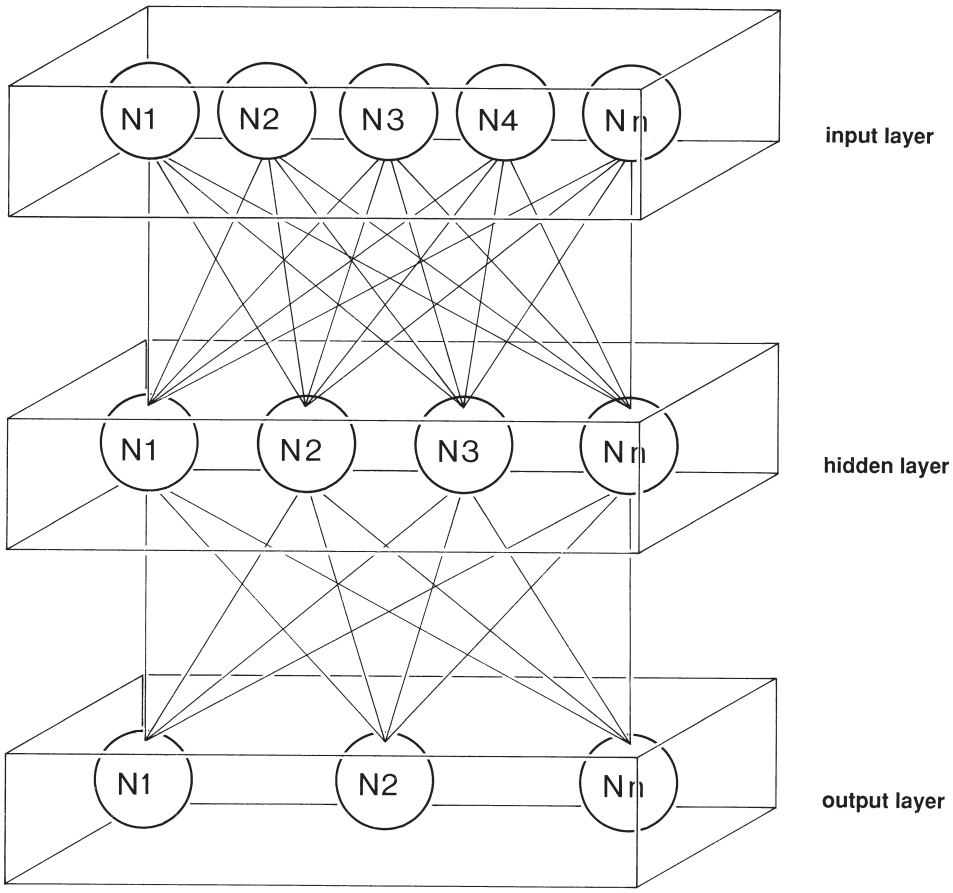

Fig. 40. Neuron and layer structure of a neural network. Each neuron (N) in one layer is connected to each neuron of another layer. The top neuron level represents the input layer, the middle level the hidden layer and the lower level the output layer.

layer act as an intermediate between the input neurons and the output neurons. They have no direct connection to the outside world: they only receive internal signals and are invisible for both the application developer and the user. Each connection between two neurons has its own specific weight, a numeric value. A weight represents the strength of the signal and, therefore, the amount by which the neuron that receives this signal is activated. Weights can have a negative or a positive value, or no value at all. A negative weight has an inhibitory influence on the activity of a receiving neuron, a zero weight means that there is no connection. The connections and their weights are of crucial importance for the networks functionality because they contain its knowledge. The collection of all weights and all connections is stored as a data matrix instead of by means of rules or other explicit representation formalisms.

Since all connections may deliver an input signal to a neuron, it receives numerous values (weights). These values are summed or otherwise calculated into a total value. Before the neuron sends its output signal it compares its total value with a so-called *threshold value*. This represents a point on which it is decided whether the neuron will actually be activated to fire its output signal or not. In other words, the activation of a neuron is determined by an *activation* or *transfer* function. As long as a neuron receives signals that do not exceed this threshold value, the neuron will not be activated. There are various types of functions, but the *step-transfer* and the *sigmoid transfer* function are most common. A step-transfer function means that only at one particular value the neuron is allowed to pass its signal. A sigmoid transfer function means there are two limits, a negative (−1) and a positive value (+1). All values in between these two limits may indeed activate the neuron.

Many different types of networks can be built. It is not only possible to vary the number of layers or the transfer function, but it is also possible to specify to which neurons the signals will be passed. The neurons in different layers may send signals to those of other layers in a forward and a backward direction, and the neurons of one layer may send signals to other neurons in the same layer or even to their selves.

Despite the fact that neural networks resemble our brain structure and that they operate faster than expert systems, their performance is yet hardly comparable to that of humans. It is said that "*... networks in fact form only the crudest approximation to networks of real neurons...*" (Copeland 1993: 222). Major discrepancies are, for instance, that artificial networks have less complex neuron connections and that they are still much more slower than our brain. The latter has to do with the way the signals are processed. In our brain all neurons can be active simultaneously because they are massively connected. An artificial network may seem to process various values simultaneously (parallel), but in fact this is still carried out sequentially because computers are basically sequentially operating devices (Carling 1992: 11-12). Another important difference is the way signals are transmitted. Computers use electronic signal transmission, while our brain employs chemical processes. In contrast to the other aspects this is clearly in the advantage of artificial neural networks, because electronic signals are more than one million time as fast as the biological method (*e.g.* Vuik 1993: 188; Patterson 1996: 28). This implies that artificial networks have the potency to process knowledge much faster than humans. However, they will never beat us as long as their electronic signals cannot be processed *en masse*.

6.3 Development process

6.3.1 INTRODUCTion

The performance and reliability of a neural network depends on many factors, such as the network structure, the quality of the training set, the training method, and the tolerated error rate. As there are no development guidelines that guarantee good results, the development process may be complicated. The best applications are achieved by experimentation, *i.e.* by means of a *trial-and-error* process in which various options are tried.

That is why the development process of a neural network is an iterative process. In this, four main activities can be distinguished. The first activity is to define the task that will be simulated and to collect sufficient data for the learning process. The acquisition of this data may be less time-consuming than the knowledge elicitation that is required for an expert system application, because there is no need to make the knowledge explicit before it can be implemented into a network. But, it is of crucial importance that the collected examples are representative descriptions of the various situations (and the associated solutions) that have been experienced within the problem domain. The learning set must cover the range of variability which the real world exhibits and must be large enough to be representative for the population about which the network application must be able to make predictions. In general, the predictions of a neural network are more reliable when it is used for *interpolation* (the new situations are similar to the training examples) instead of for *extrapolation* (situations that deviate from the training examples) (Stehouwer 1997: 66).

Designing the network structure is the second main activity in the development process. It must be decided how many layers, and neurons are required to represent the data and which transfer function and signal direction suits the task and data best. Subsequently the training method is chosen and the training can be started. It is during the training process that the neural network deduces the relevant relations between the input and output data from the examples. While an expert system needs explicit knowledge, like decision rules that are composed by a developer, the network builds its own 'decision rules'. By means of predefined mathematical functions the network software autonomously determines the connections between the neurons and their weights.

It is often during training that various problems arise and it may be necessary to edit or expand the training set or to adapt the network structure and its functions (see paragraph 6.3.3). Each accommodation is followed by a new training procedure in order to verify its effects. The fourth step in the development process is to test the network with a new data set in order to assess its generalization capacity. Again it may turn out that the network does not yet perform acceptably. In that case the training set or the network structure may need to be adapted and the training procedure must be carried out again. The knowledge representation and the training procedure will be looked at more closely in the following paragraphs.

6.3.2 KNOWLEDGE REPRESENTATION

Since the knowledge that a neural network represents is included in the training examples, it is important to obtain the right examples. Not only should these examples cover the variance that occurs in the real world, they should also be translatable into a format that the neural network can understand. Whereas knowledge may consist of images, symbols or numbers, a neural network only understands figures. Moreover, these figures must fall within the range of the transfer functions, for instance between +1 and −1. Consequently all non-numerical data must be transformed into a numerical format.

The difficulty of encoding data is that the figures must be meaningful and that they must represent the initial meaning of the non-numerical variables. For example non-numerical attributes such as 'small', 'round' and 'red' can be translated into absence and presence values, 1 and 0 respectively. Images can be encoded by representing a particular location of the image by a particular neuron and subsequently translating the grey tone or colour of that location into a numeric

value. When the data exists of both non-numerical and numerical values, such as measurements, colour indications or decoration patterns, it takes slightly more effort to create a meaningful encoding system. It may be helpful to use the data translation facilities that some commercial neural network packages offer.

6.3.3 A MATTER OF TRAINING AND EXPERIMENTATION

A neural network obtains its functionality by generalizing from examples. Therefore, the learning process implies that a large set of input and output patterns are presented to the network. The input attributes of these patterns correspond with the network's input neurons; the output attributes with the output neurons. Subsequently, the network uses the examples to construct the connections and the weights between the neurons of its input and output layer. For instance, the input attributes for a network that has to recognize fruit would be 'red, round, small' and the output 'cherry'. The configuration of the connections between the neurons and of their weights is based on calculations. It is established through a process of rehearsing examples.

In the case of a neural network, 'learning' therefore means that the connections, the connection weights, the activation functions, or a combination of these are adjusted until the error on the training examples is minimized.

Due to the autonomous nature of the learning process of a neural network the influence of a network developer is limited. During the training process the progression can be read from the height of the *mean square error* (MSE). This is the cumulative error over all training examples. The lower this error the better the network is trained. The difficulty, however, is that the MSE will almost never reach zero. Nevertheless, there are a number of network parameters and elements that the network developer can manipulate. First of all he should select a MSE that is both in reach and acceptable. Moreover, he does not have to wait until all the examples have been learned, but he still can stop the training when the acceptable MSE has been reached (*e.g.* Patterson 1996: 213). An additional advantage of stopping the training process at an early stage is it may prevent the network from overtraining itself. Overtraining means that a network has not modelled the relations within the data, but has memorized the examples. The consequence of this is that it will be poor in generalizing: it may not perform well when it is confronted with new situations.

Another element which can be used to manipulate the training procedure is the *learning algorithm*. This algorithm supervises the training process: it defines the way the errors are validated and adjusted during training. By changing the parameters of the learning algorithm the training process can be influenced. The two most frequently and successfully applied algorithms are *supervised learning* and *unsupervised learning*.

In case of a *supervised learning process* a control set of which the required output is known is used to validate the output of the training examples (Hinton 1992). It means that the network starts its training by giving an arbitrary output to a specific input of an example. It compares this output with the expected output of the example. The arbitrary outputs are of course predominantly wrong. However, the network evaluates these mistakes and subsequently adjusts the connections or the weights of the connections until the network is able to generate all outputs correctly. In other words, it 'learns' by experience. The most applied supervised learning algorithm is *back propagation*. By this way all output-mistakes are evaluated and the 'conclusions' serve as the input for the adaption of the connection weights in all previous (hidden) layers. This is meant to prevent the same error from happening again. This method is most popular for training multi-layer networks and seems to be the most suitable to practical applications (*cf.* Lawrence 1991: 76).

Unsupervised learning means that there is no control set and no feedback on the correct output. The network learns by discovering structures or clusters within the input patterns of the examples. It makes an internal representation of the data by classifying similar input patterns (Lawrence 1991: 76). One way of doing this is by means of the method of *competitive learning* (Patterson 1996: 30). In this method the weights of the neurons that respond most to a particular input signal are raised. Unsupervised learning is still one of the main research topics because its potential is not yet fully understood. It is also for this reason that it is less frequently applied to practical applications.

It may not be easy to determine a suitable learning algorithm for a particular task. There are no universal guidelines for its selection and a network developer must find out by experimentation which approach is the most appropriate. This is complicated by the fact that most software packages only allow one or a few training methods.

A third element that a developer may need to manipulate is the *training tolerance*. A network's answer that deviates slightly from the expected output may still be correct as long as the deviation is smaller than this tolerance. For instance if the tolerance is set at 0.2 and according to the training example the value of the output neuron should be 0.5, the network considers all answers between 0.3 and 0.7 to be correct.

Some neural network packages also offer the possibility to adjust the so-called *learning rate*. This parameter determines the height by which the connection strengths are adjusted during training. For instance, a learning rate of zero (0) means that the weights will not be changed at all and that the network will fail to learn. A normal learning rate is 1. It may take some experiencing, however, before an efficient learning rate is found. A learning rate that is too low will

slow down the learning process, while a high learning rate will adjust the weights too radically and this will keep the network from learning at all.
The training process is highly affected by the training data. If the training causes problems, one of the most simple methods to manipulate the training process is to start with small bits of data rather than with the entire database at once (Skapura 1996: 81). When it performs well with these small parts, it can be trained again with a larger number of training examples. Another possibility is to eliminate the examples from the training set which cause trouble and to remove internal conflicts within the training examples, such as similar inputs which have different outputs. In fact, there are various means to track down and to subsequently eliminate errors in the training set. Even conflicting examples may be coded in such a way that they can still be incorporated (*ibid.*: 80-89).
More advanced measures that may improve the training process are: removing unimportant connections in order to reduce the training time (*cf.* Hertz *et al.* 1991); manipulating the weight decay during training; adjusting the initial value of the weights; applying training algorithms that separate the examples which the network simulates correctly from those which are simulated incorrectly. The latter helps to save the correct connections and weights and to fasten the learning process (see Hertz *et al.* 1991: 158).
Usually when one employs one or a combination of these measures, it is likely that the training process can be completed successfully. This does not automatically mean, however, that the resulting network will make excellent generalizations. If, for instance, an unsuitable learning algorithm has been selected the network may turn out to be overtrained. This means that it is trained too well: it has learned the training set by 'head', but it cannot generalize from it. As a result it will not be able to interpret a new and slightly deviating input pattern.
In order to prevent this, it has been recommended to add noise to the training data, to stop the training process before the network starts memorizing them, to adjust the number of neurons and hidden layers or to try a combination of these (Patterson 1996: 206-208). Adding noise is simple and can be achieved by presenting the examples in a random order or by adding irrelevant facts. In general it is believed to be better to offer a network too much information than too little, because it is able to filter the irrelevant data but not able to think of other relevant examples.
In recent years these difficulties have been the subject of many researches and various means have been developed that may help to improve network performances. Especially in cases in which only a limited amount of data is available, there are alternative ways to minimize the *generalization error*, *i.e.* the probability of the misclassification of a new situation. One possibility to validate the accuracy of the predictions is by means of *bootstrapping*. It means that several networks are trained with the same data. The variation in the performance of the ensemble of networks enables the deduction of the average generalization error and of confidence intervals (*e.g.* Heskes, in press). Consequently, the probabilities of the output can be defined.
As the initial values of the weights may influence the generalization capability of the final network as well, it may also be worthwhile to train several networks with different weight initializations (Stehouwer 1997: 69). The different networks can then be used to obtain an average of the output of all networks. This average is a more reliable generalization than one answer from one network.
Furthermore, the validation of the network depends on the composition of the test examples. Both the training set and the test set represent only a part of the examples which may occur in the real-world situation, the so-called *population*. The newly interpreted patterns belong to that population as well, but may not have been part of the training set. When the network is confronted with deviating situations it is forced to extrapolate. As the extrapolated generalizations are less reliable than those resulting from interpolation (*cf.* Skapura 1996: 86), the average generalization error will increase. When the number of learning examples is limited, it is therefore advisable not to hold out a percentage for testing purposes. In such a case it is recommended to divide the data into segments and to train different networks with different segments. For each network a different segment can be used as a test set (*ibid.*: 87). The average generalization error of the ensemble is a better indication of the achievements than the generalization error of one network.
These are only some of a large variety of measures which can be taken to improve the reliability of the output of networks that are used for classification purposes (*cf.* Stehouwer 1997). For most training or generalization problems several expedients can be applied. However, it may take a considerable amount of patience, inventiveness, (computing) time and knowledge of both statistics and neural network technology to obtain an optimally operating application.

6.3.4 A SESSION

A session with a trained network is often a simple and quick procedure. The application comprises of a screen on which the input and output neurons are represented by means of figures, images or symbols. The latter has been used for WARP (see fig. 43). A user simply points out the attributes that represent the input data of the situation, for instance by clicking with a mouse on the input symbols. In reference to the fruit example in the previous paragraph, the user would click on the symbols that represent the attributes 'red', 'round', and 'small'. This instantly activates these neurons to

send a signal to the hidden layer. Since each input neuron is connected with each hidden neuron, the latter receives several signals. Each of these incoming signals has a different weight, because they derive from various connections. By summing the incoming signals the hidden neuron calculates its activation strength. In their turn, the hidden neurons pass a signal on to the neurons of the output layer. Again, the activation level of the neurons of the output layer is determined by the strength of the signals from the hidden neurons.
The final combination of the activated output neurons represents the network's interpretation of the information that was presented by the user. The output may be presented by means of figures, images or symbols. In contrast with the output of most expert systems, the output of neural networks represents objectively calculated probabilities. This means that if it is based on a balanced training set, the network application can give reliable indications of the chance that the output is correct. For some tasks this may be of crucial importance.

6.4 The prototype

6.4.1 INTRODUCTION

Various software packages are available for the purpose of building neural networks. This implies that there are also numerous network types. Most of them are based on the multi-layer feed-forward model, use a sigmoid transfer function and have been trained by means of the back-propagation learning method (Patterson 1996: 141). This combination has proved to be suitable to various kinds of applications. The choice of the network, however, must always be made on the basis of the characteristics of the subject that the network will be deployed for, of the data and of the desired functionality. For instance, a network that is trained by means of the back-propagation learning method will be able to generalize better than one that was trained by means of unsupervised learning. If, by contrast, one is more interested in a network that can associate rather than classify, it is recommendable to apply just a self-organizing learning method (*e.g.* Patterson 1996: 35; Lawrence 1991: 79).
At the time the prototype had to be built, there was only a limited number of software packages available that could run on an ordinary personal computer. The package that was used to develop WARP is *Brainmaker* (version 2.3).[4] This is a non-Windows[5] program which is a multi-layer feed-forward type of network. It allows a variety of transfer functions and employs the back-propagation learning method. This type was selected because its characteristics were expected to suit our purpose and because its hardware requirements were modest. In fact, it was one of the most sophisticated and user-friendly programs that would still run on a regular stand-alone computer.
It must be stressed that WARP is a limited and simple prototype which was developed to relate wear traces to contact materials only. It focuses on what in chapter 5 was called the *analysis procedure* and has merely been trained to recognize polishes. The other three wear categories (edge retouch, edge rounding and striations) have not been included yet, nor is it equipped to relate the wear patterns to motions. The reason for this limitation is threefold. First of all, WARP was merely meant as a small case-study in which the abilities of a neural network approach would be investigated for the analysis of use-wear traces. For this it was not necessary to build a large network. Moreover, it was assumed that it would be more easy to develop and train a small network rather than a large and complex one. It is known that it is best to start with a small network, especially if the data consists of conflicting examples (*cf.* Skapura 1996: 81). As this is the case with the reference collection on which WARP had to be trained, it was thought to be more practical to gradually expand a well trained small network than trying to train a large network on all data at once.
Secondly, the knowledge in WARP had to resemble that of the preliminary version of WAVES in order to enable a meaningful comparison. At the time that the development of WARP started (1992), merely the polish interpretation function had been worked out in the expert system application.
The third reason for concentrating on the traces of polish was that the reference collection yielded the largest number of training examples for this wear category. Of the 300 experiments 53.3% (160) had shown traces of polish, while edge retouch, rounding and striations had been observed on less implements: 43.6% (131), 51.6% (155) and 14% (42) respectively. Moreover, the features of the polishes had been documented in more detail (more attributes) than those of the other wear categories.

6.4.2 THE KNOWLEDGE REPRESENTED

One of the differences between the knowledge representation process for WAVES and WARP is that the knowledge had to be translated into a different format: for the expert system into decision rules and for the network into presence (1) and absence (0) scores. Another difference is that the training set for WARP consisted of unaltered examples, whereas the data was analyzed and modelled before it was implemented in WAVES. A similarity is that, based on the information that was gained from the knowledge analysis and modelling that had been carried out for WAVES, only those variables were selected for WARP of which it had been demonstrated that they relate to the applied contact material. It would have complicated the training process needlessly if irrelevant variables had been included. As a result, only five variables were selected, *i.e.*, the distribution, texture, brightness,

	Input neurons					Output neurons
	distribution abcdefghi	texture abc	brightness abc	topography abcdefgh	width abcdefgh	contact material abcdefghijklmno
exp.						
92 (fresh hide)	000001000	001	010	00010000	10000000	010000000000000
283 (silic. plants)	000010000	100	100	10000000	10000000	000000000000010
273 (cereals)	000001000	100	100	10000000	00001000	000000001000000
etc.						

Fig. 41. The training examples have been translated into series of zero's and one's before they could be presented to the network.

Architecture	
type of network:	multi-layer feedforward
number of layers:	3
number of input neurons:	31
number of hidden neurons:	31
number of output neurons:	15
number of weights in the hidden layer:	992
number of weights in the output layer:	480
activation or transfer function:	sigmoid
Training	
learning method:	supervised (back propagation)
training tolerance:	0.4
training stopped at:	650 runs (one hour)
noise:	yes
test mistakes:	25%

Fig. 42. Basics of WARP.

topography and width of the polish (see appendix III). Together these variables consisted of 31 attributes.
Since WARP was meant to interpret wear traces, the input neurons represented the 31 wear attributes, the output neurons the contact materials.[6] Fifteen contact materials were selected (see appendix III). Shell, tooth, polish '23' and polish '10' were not included because there was insufficient quantitative data on the traces that these contact materials cause. For the same reason they had not been included in the first version of WAVES either (see paragraph 5.4.2).
In order to adapt the data to the numerical character of a neural network, the wear attributes were encoded by means of absence and presence scores (see paragraph 6.3.1). This means that each neuron representing an attribute that is present in a training example receives a 1, while those that represent the attributes that are absent receive a 0. Furthermore, each neuron that represents a contact material which is responsible for the observed traces receives a 1 as well, all others a 0 (fig. 41). Consequently, each training example was represented by 46 neurons but only 6 of these had a positive value (1).
Like with the output of the expert system, several of the output neurons can be active simultaneously if the analyzed traces point at several contact materials. However, the activity strength of these neurons can differ, which gives an indication of the certainty of the interpretation. An output cell may be filled for 100 percent, but it may also be partly filled (fig. 43). The cell that is filled most represents the answer which has the highest probability.
With reference to the other network features, it was decided to use an equal number of hidden neurons and input neurons, and a sigmoid transfer-function (fig. 42). The guideline that was followed in defining the number of hidden neurons is to use between 0.5 and 2.5 times the number of input neurons (Lawrence & Lawrence 1990: 7-9). Research has shown that networks with more than one hidden layer do not necessarily perform better. Moreover, networks with many hidden neurons do not perform better than those with an equal number of

hidden and input neurons. On the contrary, too many hidden neurons may create a network that has learned the examples by heart. Such a network will not be able to generalize and to interpret slightly deviating situations (see paragraph 6.3.3).
Nevertheless, the selection of 31 hidden neurons was a guess rather than a decision based on experience. It had to prove its validity during the training and testing. If necessary, the network could be expanded with layers or neurons or be based on another transfer function. The initial value of the connection weights could not be influenced. The software package assigned these automatically.

6.4.3 THE TRAINING PROCESS

Usually, a software package provides a standard learning method. The *Brainmaker* program applied the back-propagation approach. It automatically reserves 10% (16) of the training examples for the purpose of testing the trained network and presents the training examples in a random order. Furthermore, it offers statistics (histograms) that give an impression of the training progression and of the health of the network. It allows to pause on each incorrect output in order to analyze them. Subsequently, these 'bad examples' can be written to a file in order to analyze them in more detail.
Since the training set consisted of only 160 examples, the network was expected to 'learn' these examples relatively quickly. Only if a network is trained with many more examples or if the data is very complex, it may take hours or days of training. In the case of WARP, however, it soon turned out that the network had indeed some difficulty to learn the examples. After it had been training for more than one hour (650 runs through all examples) the network was not able to learn circa 38 of the examples (26.4%) and to further improve its achievements.
Some difficulty had been expected because repeated attempts confirmed that the data contained various contradictory facts on which it would be difficult to train the network. For instance, during the knowledge analysis and modelling procedure for WAVES, it had been experienced that the reference collection contained both facts which have a similar input but a different output and facts which have a different input but an identical output (chapter 5.2.3 and 5.2.4). This means that in one case WARP must learn that the wear features (attributes) A, B and C relate to contact material X, while in another example these traces relate to material Y. It is for a neural network even more confusing that material X can also be associated with attributes D, E and F, while this does not hold for material Y.
Successively, the training tolerance, the network structure, and the transfer function were changed in order to achieve improved results. Unfortunately, none of these adjustments lead to a significant improvement of the training process. It simply refused to learn a fourth part of the examples. Subsequently the facts which the network failed to learn were analyzed. It turned out that the network was more rigorous than necessary, because many answers were not really false. In one training round I counted only 21 mistakes (14.6%) instead of 38, the number of mistakes that the network counted. Most of the differences resulted from the fact that the answers were not very persuasive. Consequently, many of them had a score that just misfitted the training tolerance. Still, most answers corresponded with the expected answer and pointed at the right contact material. In fact, all 21 'mistakes' included the right contact material in the answer. Moreover, eight of the 'mistakes' are disputable because they consisted of two outputs with equal scores on similar materials, like cereals and siliceous plants. Given the fact that such answers would have been considered correct if they would be given by WAVES, I believe it to be justified to accept them of WARP as well. In fact, it is typical for use-wear patterns that some are not absolutely diagnostic for one contact material.
This consideration reduced the number of mistakes to 13 (9%). Since this was considered to be an acceptable learning error for this prototype, the training procedure was stopped. Subsequently, WARP was tested with the 16 randomly selected examples that the program had reserved for this purpose. It concluded that 10 (62.5%) of these generated a false output. However, when the incorrect answers were assessed according to the same procedure as the mistakes that were encountered during the training process, exactly the same conclusions could be drawn on these 'mistakes'. Again, the network considered six answers (37.5%) wrong, but regarding my standards only four (25%) were really false.
It is expected that the network's achievements will improve most when the training set is improved. According to Patterson accurate learning occurs when only the desired examples are presented and the non-desired examples are excluded (Patterson 1996: 26). This means that the problematic examples will have to be excluded from the training set. The difficulty with this, however, is that it cannot be known with certainty that the examples which generate incorrect answers are the ones the network cannot learn. In each round of training different examples may cause problems because the network adapts itself constantly to the encountered mistakes. Consequently, an example that was learned well in one round might generate an incorrect output in the next.
In our case, the best way to trace conflicting examples is probably by training different networks on small subsets. The subsets that render the largest training errors can then be analyzed and disposed of the problematic items (Skapura 1996: 81).

However, it must be questioned whether the resulting knowledge base would still be representative of the real-world situation. If all the 'problematic examples' — which are likely to occur in practice — are discarded, it is to be expected that WARP will not be able to generalize well. Moreover, the exclusion of the problematic wear patterns would have meant that WARP was no longer comparable with WAVES, because it would not have been based on the same knowledge.
Based on these considerations and on the obtained achievements, it was decided that the network had been sufficiently trained to suit our purpose, *i.e.*, a comparability study. Since WAVES had not been optimized either, the network was believed to perform comparably. An additional argument for this decision was that the network's learning statistics looked good. It showed a neat bell-shape curve, which is an indication for a healthy pattern of connections and weights (Lawrence & Lawrence 1990: 5-10).

6.5 An assessment of the prototype

6.5.1 INTRODUCTION

Compared with WAVES, the development of the neural network prototype was more easy and less time-consuming. One of the reasons for this is that WARP was only meant as an experiment rather than a practically functional application. Consequently, the demands concerning the knowledge handling and user interaction that had to be taken into account during the design and implementation of WAVES (chapter 5.3), did not play a dominant role during the development of WARP.
Nevertheless, the latter has been assessed on the same aspects as WAVES (see paragraph 5.8) in order to give an impression of the similarities and differences between the two applications and, in particular, between a neural network and an expert system approach in general. In outline, the prototype will first be evaluated against the demands that *computer archaeologists* (paragraph 6.5.2) and use-wear analysts (6.5.3) pose on knowledge-based systems. Secondly some suggestions for the improvement of the prototype will be discussed.

6.5.2 ANSWERS TO EXPECTATIONS OF *COMPUTER ARCHAEOLOGISTS*[7]

It was stated in previous sections (paragraph 2.4.1 and 5.8.2) that the most important complaints from archaeologists in relation to knowledge-based applications are that they offer a single answer without leaving an opportunity for uncertainty, that they are 'black boxes', that they fossilize knowledge, and that they lack user friendliness. In the design of WARP some complaints could easily be met, but others turned out to be more difficult to meet.
The demand to consider uncertainty or alternatives in the application's answer is easy to incorporate in a neural network. In fact, neural networks cannot exclude uncertainty. Their output is the outcome of a statistical estimation. In contrast with most expert systems these estimations represent probabilities which are calculated on the basis of quantitative data rather than subjective impressions. Moreover, the reliability of a neural network can be deduced by calculating the confidence intervals of its generalizations (*e.g.* Heskes, in press).
The second major concern of archaeologists, the *black box* problem, is more difficult to overcome with neural networks than with expert systems. They are more or less composed as a black box: some input goes in, some output comes out, and the intermediate actions are invisible to the user. This is due to the fact that they have not been developed to simulate heuristic reasoning but to handle a large number of figures. However, the black box characteristic is one of the aspects that is the subject of research. Some facilities have already been developed that enable the deduction of 'rules' from the data structure within a neural network.[8] Moreover, this lack of transparency can also be overcome by combining a neural network application with a program that serves as a communication interface.
With respect to the concern of knowledge fossilization, it was argued previously (paragraph 5.8.2) that I consider this aspect to be inherent to the recording of the results of scientific research, irrespective whether this is done in a book or a computer program. It can be avoided by making sure that the involved knowledge is evaluated and adjusted periodically. Obviously, this also applies to the knowledge in a neural network application. The difference, however, with an expert system is that updating is more easily effectuated in a neural network. Whereas an expert system requires a complete, and therefore costly, evaluation of the knowledge rules etc., a neural network can simply be trained again with a revised database.
Finally, it was stated that archaeologists prefer user-friendly and easily understandable applications. Most neural networks, like WARP, are indeed easy to handle and to understand. It was shown in paragraph 6.3.4 that a session consists of very simple acts. A user only has to click on the five input neurons that correspond with the characteristics of the observed use-wear traces (fig. 43). The network then compares this information with its reference matrix, calculates the strength of the output neurons and lightens the output neurons that represent the contact material that the network associates with the presented wear traces. Due to the fact that it consists of compiled data matrices only, it takes the network less than a second to generate its answer. A session with WAVES will take a student at least a quarter of an hour. This means that for tasks in which time is a crucial element, the choice between an expert system and a neural network should be in favour of the latter.

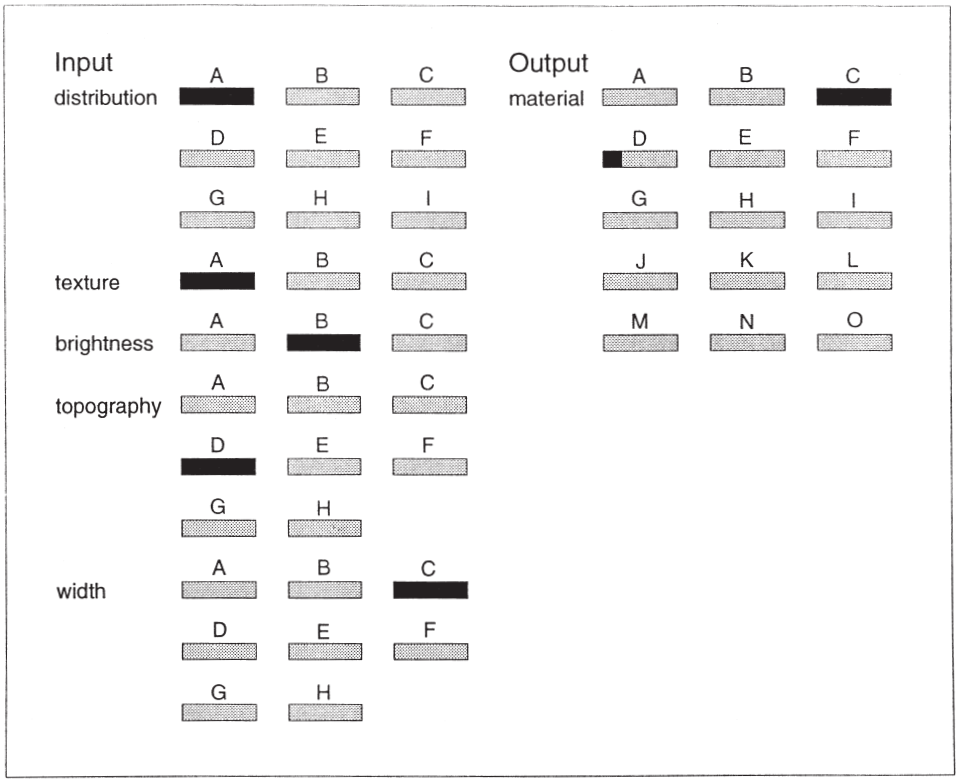

Fig. 43. The input/output screen of WARP. The left hand side shows the input neurons, the other side the output neurons. The user chooses the five input neurons that correspond with the observed wear traces and the system shows the contact materials that may have caused that wear pattern.

To summarize this paragraph, neural networks can meet several of the demands of archaeologists. They are able to provide alternative answers, they can give an indication of the reliability of their output. Once they have been trained, they are easy to employ by laymen and they operate very fast. But, they are not meant to explain their answers and for archaeology students the internal reasoning processes may be difficult to understand.

6.5.3 ANSWERS TO EXPECTATIONS OF USE-WEAR ANALYSTS

Apart from the more general demands that computer archaeologists pose on knowledge-based applications there are also more specific demands. These are posed by use-wear analysts. In paragraph 4.5.6 it was stated that use-wear analysts expect from automated approaches that they provide a means for standardizing the method of analysis and for formalizing the knowledge that is involved with it, that they incorporate all wear categories, that they support the learning process of students, that they are maintenance friendly, and that they are applicable to different archaeological assemblages. Like with the more general demands of the computer archaeologists, neural networks, and WARP in particular, have turned out to be able to meet some of these specifications remarkably well, but others less.

First of all, neural networks indeed provide a means to formalize and model the knowledge that is involved in wear trace analysis and offer a standardized approach to the method. The use of an application like WARP implies that the method of analysis will yield less subjective results. Therefore, it improves the comparability of the interpretations of different analysts. Moreover, it may help to trace deficiencies within the knowledge or may be used for other research purposes. For instance, a professional use-wear analyst may employ WARP in studying the diagnostic value of specific attributes: by manipulating the input it may become clear what the effects are on the output. The neural network approach can also be used to ground interpretations with statistical reliability.

With regard to the incorporation of all aspects of use-wear traces, this will not be very easy to accomplish within WARP. Presumably the database contains too many contradictory facts to allow for a smoothly training process. Moreover, if the network would have to include all wear categories, both the number of input neurons and the training time would increase drastically. This would also mean that additional computational facilities would be demanded, like a more powerful computer, but also that the reference collection would have to be expanded. The present training set

contains insufficient examples for training such a large and complex network. This does not mean, however, that the neural network approach is by definition unsuitable to incorporate all wear categories. If it is unfeasible to develop one large network a good alternative is to build separate networks for all wear categories.
With reference to the demand that a knowledge-based approach ought to support and preferably even quicken the learning process of students, an application like WARP is less useful than an expert system like WAVES. It is not very well equipped to teach the principles of the analysis process because it does not represent its knowledge in a way that is understandable for archaeology students. The data matrices are composed for internal calculation purposes, not for making the reasoning processes accessible to others. Moreover WARP does not offer a sophisticated user-interface with illustrations, explanations or instructions, it does not offer the analysis process as a step-by-step procedure, and it cannot react when a student makes input mistakes.
On the other hand, the demand of maintenance friendliness can be easily met by a neural network. It only has to be trained again with a revised database. Even an adaptation of its structure can be accomplished more easily than that of an expert system application. When the number of neurons has to be changed, one has to try and find a new balanced structure by going through a new process of trial-and-error experimentation.
With respect to the demand that an automated approach for use-wear analysis must be applicable to various archaeological assemblages, there is no large difference between WAVES and WARP, because this aspect relates to the range and quality of the involved domain knowledge rather than to the way it is represented and handled. Since both applications are based on the same knowledge, they can be applied to the same range of assemblages (paragraph 5.8.3). Nevertheless, WARP may turn out to perform better in situations that demand interpolations, while WAVES may be better in extrapolations.
To summarize, the neural network approach can be used to standardize use-wear analysis and to formalize the involved knowledge, it is maintenance friendly and very well applicable to interpret the traces from various archaeological assemblages. However, it does not suit educational purposes very well because it does not offer means to validate its answers, and its autonomous knowledge handling approach does not contribute to a better understanding of the involved knowledge.

6.5.4 SUGGESTIONS FOR ADDITIONS

In the previous paragraphs it has been argued why the present form of WARP is not very suitable for educational purposes. In order to exploit the power of the neural network approach for the benefit of use-wear analysis, it is advisable to apply WARP for different purposes, like research on the diagnostic value of wear traces. Still it would need some improvements in order to obtain better interpretations.
First of all, it needs more training. With many of the contact materials too few experiments have been done to allow for reliable statistical inferences. Scientifically, it would also be interesting to analyze the facts that it failed to learn in more detail. Moreover, it could be tried to train WARP on the altered data that was used to build the knowledge rules for WAVES instead of on the raw data from the reference collection. This would probably decrease the number of examples that cannot be learned, because the altered data contains less conflicting wear patterns.
Furthermore, WARP should be supplemented with the other wear categories, with a possibility to interpret applied motions, and perhaps with additional interpretational means such as residue analysis, or with a facility to validate hypotheses. An addition of the other wear categories may best be achieved by dividing the application into small networks which each handle one category. These small networks are more easy to train and can eventually be brought together into one application.
With reference to the range of the conceptual knowledge of the application, the scope of the experiments would need some expansion as well, especially because WARP has only been trained on examples from the experimental programme. Unlike WAVES, it has not been supplemented with expert knowledge. Moreover, as it was recommended for WAVES, it would have to be expanded with knowledge or experimental examples from multiple experts.
An additional problem, to which no adequate answer has been found in the design of WAVES ánd WARP, is the subjective nature of the input the system receives from the user. Whereas in designing WAVES these difficulties were partly met by offering illustrations and definitions of the variables and attributes, this is no solution for WARP because these cannot be incorporated using Brainmaker.
Finally, WARP shows some imperfections that are present in WAVES as well. For instance, both applications do not differentiate between the traces on the ventral and dorsal face of the implements. While this is complex with WAVES, this could be rather easily adjusted in WARP. It would simply imply a training procedure with twice as many input neurons. The main provision, however, is that this kind of information would be supplied by the reference collection.
To conclude, several of the difficulties and hiatuses that were encountered during the development of WARP are both inherent to the applied approach and to the characteristics of the method of analysis and the available data.[9] Especially the limited amount of data makes it difficult to simulate use-wear analysis by means of a neural network. Nevertheless, it can be a useful approach to standardize this

method of analysis and to formalize its knowledge. In fact, they can be a valuable extension of conventional statistical techniques. However, it is an oversimplified conclusion to state that, in comparison with expert systems, neural networks are superior regarding their functionality and social acceptability. Whether one approach is better than the other depends entirely on the aim of an application and on the type of data that it involves.

notes

1 Since the beginning of the nineties the research on neural network technology flourishes and the technical developments in this area succeded each other speedily. The application that is presented in this chapter was built in 1992 and, therefore, does not reflect the latest technical state of affairs.

2 The more recently developed neural network methods are also referred to as *parallel distributed processing* (PDP).

3 This paragraph does not give a comprehensive view of the technical possibilities of neural networks. It merely is a simple introduction to the elements that a neural network comprises. There are many good books on this subject, to which the interested reader is referred to.

4 Brainmaker is a registered trademark of California Scientific Software.

5 Windows is a registered trademark of Microsoft Corporation.

6 In the guide book that accompanied the software package it was recommended to use between 50 and 200 training examples for a network with a limited complexity. It did not mention the optimal relation between the number of training examples and the number of neurons (and weights).

7 *Computer archaeologists* refers to people who are specialized in applying and developing quantitative and automated methods for archaeological issues.

8 Pers. comm. prof. dr. J.N. Kok, Department of Computer Science, Leiden University.

9 With reference to the achievements of WARP it must be stressed that I merely explored the possibilities of one type of neural network, while more than 20 different types exist. Consequently, another type of network may have yielded different results.