Abstract
In this study neural network expert systems are investigated. The article started with simple description of neural networks and this is followed by the comparison of rule-based and neural network systems. Examples for neural decision systems and neural experts systems are given and the difference between neural decision and expert systems is emphasized. The construction of knowledge the base, forward and backward chaining within neural expert systems, rule extraction from knowledge base and debugging the neural expert system are explained and supported with examples from MACIE.

1. Introduction
Networks that mimic behavior of neurons are called neural networks. The study of computation with neural networks goes back to McCulloch and Pitts (Mehrotra et al., 1997). Attention of AI community is directed to neural networks after to the work of Rumelhart and McClelland, “Parallel Distributed Processing: Explorations in the Microstructure of Cognition” (Mehrotra et al., 1997). They showed that neural networks have many properties, which are absent or hard to implement with von Neumann machines, such as parallel computation, content addressable memory, coarse coding, graceful degregation etc. Besides, neural networks can achieve many intelligent tasks easily, such as learning from examples, pattern recognition, dealing with noisy environments etc.

From the point of view of expert systems, neural networks are important mainly because of their learning abilities. In expert system development, construction of knowledge base is the most time and money consuming task. Neural networks are considered as devices that can minimize this charge by acquiring necessary knowledge from examples of the domain. Expert systems with neural network knowledge base are called neural expert systems. In this review, I would like to report some literature that gives examples for neural expert systems and for their distinctive features, such as construction of knowledge base and extracting rules form a knowledge base. But let us start with explaining briefly what a neural network is.

2. Neural Networks
A neural network consists of a set of nodes and links between nodes. Nodes have ability to take an input, execute some function and give an output (activation). Activation of a node follows a path defined by links and become an input for another node. Links between nodes are weighted. Therefore, effect of one node on another is defined by the strength of the weight between these nodes. Each node
takes input from (and gives output to) many other nodes. Group of nodes that are connected to others in a similar way are called layer of nodes. The simplest neural network architecture has two layers: input and output.

A group of two layered networks are universal computers because each of them is capable of implementing ‘and’, ‘or’ and ‘not’ gates (Mehrotra et. al., 1997). On the other hand, two layered neural networks can implement only linearly separable functions. For functions that are not linearly separable, we need at least three layers: input, hidden and output. It was shown that a three layered network is a universal computer (Mehrotra et. al., 1997).

The most important property of neural networks is their learning ability. Learning is achieved by exposing a network to a set of training examples and executing a learning algorithm. There are many different learning algorithms for neural networks. Some of them provide supervised learning (learning with a teacher) others provide unsupervised learning. Supervised learning is based on calculating the difference between expected output and observed output and changing weights of the network to obtain the desired output. Unsupervised learning is based on changing weights by only exposing to an environment without a teacher. We do not want to give details of supervised and unsupervised algorithms because there are many texts which provide explanation for such issues (Mehrotra et. al., 1997).

3. Rule-Based Systems vs. Neural Network Systems

Before getting into details of the structure and function of a neural knowledge base lets compare rule-based systems with neural network systems. We would like to compare the inherent strengths and weaknesses of each system.

Rule-based systems have an excellent explanation capability because the trace of the solution can be built up from the list of applied rules. Neural networks, on the other hand, are black boxes. It is hard to trace some path following the propagation of activation within the network because neural networks are parallel computers. Therefore, neural network systems have little or no explanation capacity.

Rule based systems require an expert to learn from the heuristic rules of the domain. Average development time of rule based systems is from 12 to 18 months. On the other hand neural networks require set of examples. From those examples neural networks are able to extract necessary relations between data without help. It takes few weeks or months to develop a neural network system. The disadvantage of neural network systems is that they need many examples. Training a network with small set of examples is useless.

Many rule based systems are available for public reference therefore it is easy to examine their true capacities and limitations. On the other hand, today, there are a small number of neural network systems available for public reference, because firms, which developed such systems, generally do not want to share benefits of them with rivals. Rule based system is an old concept
therefore there are many development shells available for them. On the other hand, there are few shells for neural network systems.

It is very hard to develop and maintain large rule-based systems. The maintenance burden of a small network is same with a large network. Execution time does not change with respect to the size of knowledge base in neural network based systems, but it changes with respect to the size of knowledge base in rule-based systems. Rule based systems works fine on ordinary computers. But for best performance, neural networks require specifically designed chips. And lastly, neural network systems do not suit every knowledge engineering task. Classification type of tasks is suitable for neural networks.

3. Neural Decision Systems
We reviewed the benefits and costs of rule-based and neural network systems. Every rule based systems is not an expert system (Jackson, 1999). Likewise, every neural network solution to a classification problem is not a neural expert system. In this section, we would like to give an example to a neural decision system.

William G. Baxt of Department of Medicine in the University of California developed a neural decision system that performs better than any known system (including physicians) for the diagnosis of acute myocardial infraction (Baxt, 1990). The author built a multilayered neural network (20X10X10X1). Input of the network is a set of variables that represents medical condition of a patient and output of the network is the diagnosis result for the acute myocardial infraction. To train and test the network, the author used 356 cases; 236 of them did not have acute myocardial infraction and 120 of them did have infraction. He randomly divided this set into two and used one set for training the other for testing the network. After training, the network correctly identified 92 of the patients with acute myocardial infraction and 96 of the patients without infraction. Author discussed that the patient histories are vague and imprecise besides the match between symptom clusters and clinical results varies from case to case. In other words, the environment is highly noisy. It is well known that neural networks work within noisy environments successfully (Gallant, 1993).

We reviewed a successful example of a classification system which uses neural network as a knowledge base. On the other hand, Baxt’s network is not an expert system for number of reasons: it does not have an ability to make inference upon partial data; rather it requires full information about the patient. It does not interact with user to obtain further data when needed. And it cannot justify inferences by explaining them.

4. Neural Network Expert Systems
How neural experts system implement features that are lack in neural decision systems. We would like to explain this issue by examining the operations of MACIE (Matrix Controlled Interface Engine) (Gallant, 1993). MACIE is an expert system shell with a neural knowledge base. Given a set of
variables, dependency information between variables, a set of questions to elicit value for variables and a set of training examples, MACIE constructs a neural network knowledge base and through inference engine it provides an expert support. Please refer to Figure 1 for the architecture of MACIE. The user interacts with system through inference engine which controls neural knowledge base and question databases. In this section, we will explain the construction of neural knowledge base and then we will clarify the operations of inference engine.

Figure 1 The architecture of MACIE

4.1 Neural Knowledge Base

Neural network knowledge base of MACIE consists of nodes and weighted links. Nodes in MACIE represent variables in the expert domain. By this way, each node gains a semantic interpretation. As a result, activation of nodes has a natural interpretation: the existence of that variable. Links between nodes represent dependency relations between variables. Each link is weighted and weights are obtained by a supervised training algorithm.

We already mentioned in Section 2 that neural networks learn by changing weights between nodes. Generally, the architecture of the system (number and arrangement of nodes, layers and links) is given a priori. Event tough, there are examples of learning systems that build nodes, layers and links by themselves. MACIE falls into the first category. But construction of the architecture is automatic given the variables and dependency relations between those variables in the expert domain.

In MACIE, variables in the problem domain are represented with nodes. Figure 2 gives an example knowledge base for MACIE. Variables include symptoms (S1-S6), diseases (D1 & D2) and treatments (T1-T3). Dependency relations between variables are implemented with links between nodes. In Figure 2, the dependency information includes S1, S2 and S3 are connected to D1; S6, D1
and D2 are connected to T1 etc. MACIE may add some extra nodes (intermediate variables) based on task demands. We will come to the addition of intermediate variables to knowledge base after we explained how network is trained.

Figure 2 An example knowledge base

Weights of links and threshold values of nodes are assigned after training the network with a set of examples. Set of examples may include, for example, patient records such that each record mentions which symptoms co-occurred with which diseases and which treatment was applied to the disease. For example, one training case may declare that patient had symptoms S4, S5 and S6 and disease D2 and D2 was treated by T2.

The training phase starts with entering symptoms of one case to the network and reading the disease output given by the initial weight – threshold configurations. If there is no difference between expected and obtained output (or disease) than next case is processed. If there is a difference, weights and thresholds are adjusted in a way that next time error will be decreased for the same case. After the match between diseases and symptoms are learned, the network starts to learn to output treatments given diseases and symptoms. The only restriction of this training algorithm is that examples should
not include variables that are not exist in the variable list. The learning algorithm goes cases one by one and at the end the network guaranteed to learn the training set, in other words it will output correct diseases and treatments given a set of symptoms.

In section 2 we mentioned that problems that are not linearly separable cannot be solved with two layered networks. If MACIE confronts with a nonlinear association between variables, it constructs an intermediate variable. By this way, MACIE is able to map any kind functions between variables.

Apart from the neural network, knowledge base of MACIE also includes a set of questions (please refer to Figure 1). Questions are necessary for obtaining user input for each variable. Suppose that S1 is high fever. In the question database, the question associated with S1 would be “Do patient has a high fever?” The answer would be given as an activation value of the node that represents S1.

4.2 Neural Network Inference Engine
We have seen in Section 3 that neural knowledge base is not all of a neural expert system. An expert system must be able to make interferences based on partial data, able to interact with user to justify given answers. In MACIE, Neural Network Inference Engine is responsible from these tasks. Inference with partial data and prompting user for further information is achieved by forward and backward chaining mechanisms and they will be covered in this section. Justifying answers is related with rule extraction from neural knowledge base. Rule extraction will be covered in section 4.3.

The problem of making inference given partial information is solved with forward chaining strategies in MACIE. Please refer to Figure 3 for an instance of a session in MACIE. At this moment information related with some variables is available for sure (nodes with True and False activation), information related with some variables is unavailable for sure (nodes with Unknown activation), and information related with some variables may be available in future but not available at this moment (nodes with no (?) activation). For example, one may need the value of U7 at this moment because we want to make an inference about U7 or U8. Is U7 True or False at this moment? This is forward chaining with partial information. The algorithm of forward chaining is very simple:

Compute weighed sum of all available for sure and unavailable for sure data for node i (CURRENT_i). Compute sum of the absolute values of weights of data with no value (?) for node i (UNKNOWN_i). If |CURRENT_i + THRESHOLD_i| > UNKNOWN_i, than infer that node (or variable) i is True else False.

This algorithm guarantees that the value of U7 for example, would not change even if nodes with no activation (?) will gain some activation in future.
The problem of determining which data (input variable) is needed for further inference is solved by backward chaining strategies of MACIE. The first step in backward chaining is to compute confidence estimation of a variable. Confidence estimation of variable $a$ (node) $U_7$ is an estimate of the likelihood that an unknown variable $U_7$ will eventually be deduced to be true or false. There may be several different heuristics for confidence estimation. One of them is:

For a node $i$ with known activation $u_i$, confidence estimation is equal to the activation of that node, $\text{Conf}(u_i) = u_i$. For an input node $i$ with unknown activation confidence estimation is equal to zero, $\text{Conf}(u_i) = 0$. For another node $i$ with unknown activation confidence estimation is equal to weighted sum of confidence estimations of nodes $j$ connected to node $I$ divided by sum of the absolute values of weights of data with no value (??) for node $i$ (UNKNOWN$_i$), $\text{Conf}(u_i) = \Sigma w_{ij} \text{Conf}(u_j) / \text{UNKNOWN}_i$.

With this calculation confidence estimation always lie between -1 and +1. After confidence estimations are computed, the inference engine executes the backward chaining algorithm:

Figure 3: An instance of MACIE session.
Select an unknown variable (node) i with maximum confidence estimation. Then find an unknown variable j which has the greatest absolute influence on it; or find max|wij|. If j is an input variable ask user input. If not assume that variable with maximum confidence estimation is j and continue.

This algorithm guarantees that input prompted by user will assign maximum value for an unknown variable i.

We have examined how forward and backward chaining is achieved by a Neural Network Inference Engine. We would like to direct your attention to the simplicity of these algorithms.

4.3 Rule Extraction from Neural Knowledge Base
Rule extraction is important for many reasons. Neural expert system must provide some explanation facility in order to be regarded as true expert systems. Besides, it may be useful to extract rules from a neural network knowledge base while debugging or developing the system. Another possibility is to use rules of a neural network knowledge base within a conventional expert system.

There are two main criteria that rules must satisfy: validity and maximal generality. Validity refers to the fact that conclusion of a rule must be correct given values of unmentioned variables. For example rule “If U1 is true and U2 is false than U3 is true” must be valid regardless of values of U4, U5, U6 etc. Maximal generality refers to the fact that we want rule to be maximally general. In other words, if we remove any conditions the rule will no longer be valid.

There are two main issues that must be decided on before rules are extracted from a neural network expert system. Should we extract single rule for each inference? Because of the multiple connections between nodes in a neural network, the knowledge is distributed within these links. It is always possible to extract more than one rule for each variable in a neural network. The extraction of a single rule may result with obtaining a partial rule. However, extracting multiple rules may result in conflict because most probably different connections in a neural expert system compete to activate the same node. The solution would be to assign a threshold for certainty of rules and ignore rules below that value. Another issue that needs clarification for rule extraction is the depth of rule extraction. Should we express rules only in terms of nodes which directly are connected or should we limit ourselves with input variables only.

There are several algorithms for rule extraction. The basic strategy of these algorithms is to select variables that contribute the activation most. We do not give the details of algorithms. Please refer to (Gallant, 1993) for in-depth investigation of the issue.
4.4 Debugging an Neural Network Expert System

We have pointed that extraction of rules help debugging of the neural expert system. Conventional expert systems are hard to debug because interaction between rules is unknown and we may introduce a bug if we change a rule.

Debugging a neural network expert system is relatively easy, or at least guarantees an increase for performance. Figure 4 shows the debugging cycle of a neural expert system. Debugging is based on introducing new examples from the expert domain to the network and let the network learn intricacies of the expert domain. The system can be tested by running it on some test problems. The output of the system is evaluated by an expert and if the output is wrong, test case with correct output is added to the training examples. Another way to debug the neural expert system is to generate rules from knowledge base. Rules are examined by an expert and he/she will give further examples related with invalid rules. Those examples are added to the training set. By this way, the neural knowledge base becomes tuned much more precisely to the problem domain as the debugging process continues.

![Debugging cycle of neural expert systems](image)

**Figure 4 Debugging cycle of neural expert systems**

**References**


